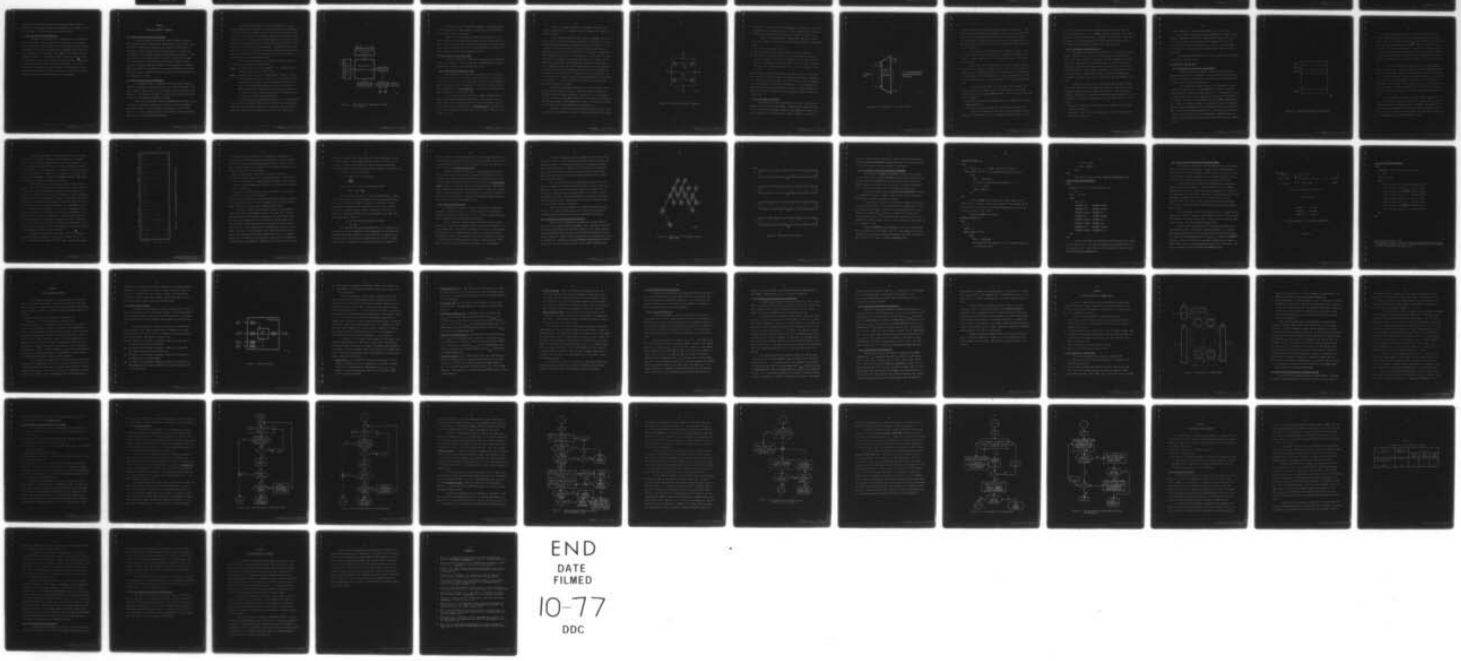
AD-A044 281 ILLINOIS UNIV AT URBANA-CHAMPAIGN COORDINATED SCIENCE LAB F/G 9/5
FAULT DIAGNOSIS OF SEMICONDUCTOR RANDOM ACCESS MEMORIES.(U)
MAY 77 S M THATTE DAAB07-72-C-0259
UNCLASSIFIED R-769 NL

1 OF 1
ADA
044281



END
DATE
FILMED
10-77
DDC

REPORT R-769 MAY, 1977 UILU-ENG 77-2216

COORDINATED SCIENCE LABORATORY

12 NW

FAULT DIAGNOSIS OF SEMICONDUCTOR RANDOM ACCESS MEMORIES

SATISH MUKUND THATTE

UNIVERSITY OF ILLINOIS - URBANA, ILLINOIS

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* (6) FAULT DIAGNOSIS OF SEMICONDUCTOR RANDOM ACCESS MEMORIES. (14)		5. TYPE OF REPORT & PERIOD COVERED (9) Technical Report, (12)
		6. PERFORMING ORG. REPORT NUMBER R-769, UILU-ENG-77-2216
7. AUTHOR(s) (10) Satish Munkund Thatte		8. CONTRACT OR GRANT NUMBER(s) (15) DAAB 07-72-C-0259
9. PERFORMING ORGANIZATION NAME AND ADDRESS Coordinated Science Laboratory University of Illinois at Urbana-Champaign Urbana, Illinois 61801		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS (12) 73 p.
11. CONTROLLING OFFICE NAME AND ADDRESS Joint Services Electronics Program (11)		12. REPORT DATE May 1977
		13. NUMBER OF PAGES 65
14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)*		15. SECURITY CLASS. *(of this report)* UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

Semiconductor Random Access Memories
Functional Testing
Pattern Sensitivity Testing

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

This report deals with the problems of testing semiconductor random access memories and of locating faults on a memory board. Memory test procedures can be divided into three classes, functional testing, pattern sensitivity testing and DC parametric testing. Existing test procedures for testing semiconductor memories are either limited in their fault coverage or require a prohibitive amount of time. A new functional test procedure based on a fault model that takes into account a large variety of faults encountered with semiconductor

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

20. ABSTRACT (continued)

memories is presented. The fault model is not based on the "gate" level as in classical fault diagnosis but is formulated on a higher level in terms of functional blocks, like the decoder and the memory cell array. The proposed functional test procedure takes $O(n \cdot \log_2 n)$ units of time where n is the number of words in memory. This gives a dramatic improvement in the testing time required over well-known test procedures line "GALPAT" and "Walking Ones" which take $O(n^2)$ units of time. Algorithms for the functional test procedure are given.

A model for pattern sensitivity caused by device anomalies and parasitic effects id eveloped in terms of various delays and dynamic timing parameters associated with memory. Test procedures based on this pattern sensitivity model are presented for detecting pattern sensitivity. These are also of $O(n \cdot \log_2 n)$ or $O(n)$.

The problem of locating faults on a memory board to memory chips, decoder logic, data registers, and bussing structure is discussed. A test scheme for this problem is given. Finally various test procedures presented in the thesis are evaluated for fault coverage, time requirement and east of implementation.

FAULT DIAGNOSIS OF SEMICONDUCTOR RANDOM ACCESS MEMORIES

BY

SATISH MUKUND THATTE

B.E. (Hons.), Birla Institute of Technology and Science, 1975

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1977

Thesis Adviser: Professor J. A. Abraham

Urbana, Illinois

UILU-ENG 77-2216

FAULT DIAGNOSIS OF SEMICONDUCTOR RANDOM ACCESS MEMORIES

by

Satish Munkund Thatte

This work was supported in part by the Joint Services Electronics Program (U.S. Army, U.S. Navy and U.S. Air Force) under Contract DAAB-07-72-C-0259.

ACKNOWLEDGMENT

I wish to express my gratitude to Professor Jacob Abraham for his guidance and encouragement. I would like to thank G. David Frye and Ira Chayut for coding and implementing the test procedures proposed in this thesis. Thanks are also due to Mrs. Mary McMillen for the immaculate typing of this thesis.

TABLE OF CONTENTS

CHAPTER Page

CHAPTER 1

INTRODUCTION

With the rapid development and advances in semiconductor technology, there has been a continuing growth of larger and denser semiconductor memories. This has posed a real challenge in the area of testing these memories. As more and more cells are packed on a semiconductor chip, not only does the number of failures associated with memory increase but the nature of failure modes becomes more complex and subtle. This makes testing a more difficult task. In addition, more time is required to test memories because of the increase in size.

mands on test procedures required for testing today's semiconductor memories are to detect faults of a wide variety and a complex nature, and to minimize the total testing time so as to make testing a cost effective proposition.

We are interested in testing semiconductor memory chips as well as memory boards housing these chips. Since it is impossible to correct a failure in a semiconductor memory chip, we are interested only in detecting a fault on the chip and not in locating it to a particular part of the chip like the cell array, decoder or Read/Write logic.

On the other hand, while testing memory boards, we are interested not only in detecting faults but also in locating them to the memory chips, decoder chip, data registers or the data and address bussing structure.

We will discuss the testing of only Random Access (Read/Write) semiconductor memories (RAMs) and memory boards housing RAMs. The testing of

Read-Only memories (ROMs) can be viewed as a special case of the testing of RAMs.

Test procedures for a memory can be conceptually divided into three parts:

1. Functional testing*: the test must detect physical failures which cause the memory to function incorrectly.
2. Pattern sensitivity testing*: even though a memory has no physical failures, there could be device anomalies and parasitic effects which could make its dynamic behavior sensitive to data patterns; the test must detect these conditions.
3. DC parametric testing: the DC parameters like power consumption, fan-out capability, noise margins, leakage currents, etc., must be checked.

These test procedures as well as the fault location procedure for a memory board are described in more detail below. The approach taken in this thesis toward testing is also pointed out. Chapters 2,3 and 4 describe the testing and location algorithms and Chapter 5 is an evaluation of the procedures.

1.1 Functional Testing

This test procedure would determine whether a memory is functional. A memory is defined to be functional if it is possible to change every cell from a 1 to a 0 as well as from a 0 to a 1, and to read every cell correctly when it stores a 1 as well as when it stores a 0, independent of the state of the remaining cells. Obviously if we want to test a memory in a reasonable amount of time, it is not possible to check each cell for all possible states

*No standard nomenclature exists in the literature for these two terms; our terminology is common.

of the remaining cells, as this would make the testing time on the order of 2^n, where n is the number of cells in the memory [1].

Therefore it is necessary to develop a functional test procedure on the basis of a fault model which is comprehensive enough to account for a majority of the failures associated with semiconductor memories. In Chapter 2, we describe various failure modes associated with semiconductor memories. Based on this discussion, a fault model is constructed for the development of a new functional test procedure. This procedure needs a testing time on the order of $n \cdot \log_2 n$, where n is the number of words in memory. The theoretical basis for this test procedure is presented and algorithms for its implementation are given. The fault model is based on only the permanent faults (as opposed to intermittent faults) such as a cell stuck at 1 or 0, shorts or capacitive couplings between cells, or decoder faults. It is important to differentiate these faults from device anomalies and parasitic effects that may give rise to what is called pattern sensitivity.

1.2 Pattern Sensitivity Testing

Dynamic timing parameters of semiconductor memories like access time, data set-up and release time, and address set-up and release time depend on the data pattern and the sequence of memory accesses. Under normal circumstances these parameters remain within the given tolerances for all data patterns and sequences of memory accesses, and hence should not cause any problem. However, various device anomalies and parasitic effects may cause a sufficient change in these parameters, which, with a change in the data pattern or sequence of memory accesses, results in a wrong memory operation. Moreover, as described in [2], under a particular combination of data

pattern and sequence of memory accesses, parasitic capacitor coupling can manifest itself in loss of data in some memory cells, due to peculiarities of design and layout of the memory chip (and not due to changes in the dynamic timing parameters).

Thus device anomalies and parasitic effects could make a memory "sensitive" to the data pattern and sequence of memory accesses. This is called pattern sensitivity. It should be emphasized again that we are talking about memory malfunction caused by device anomalies and parasitic effects and it should be distinguished from incorrect functional behavior caused by permanent faults.

The development of test procedures to identify pattern sensitivity related to the peculiarities of design and layout of the memory chip can be considered the most difficult part of testing. It involves a thorough understanding of the design and layout of the memory chip, device anomalies and parasitic effects. This information is usually not available to test designers. Moreover, in a discussion like this, we cannot treat the idiosyncrasies of a particular design or the effects of design and layout errors.

Therefore a model for pattern sensitivity in terms of only the dynamic timing parameters is proposed in Chapter 3. A test procedure based on this model is described for detecting pattern sensitivity

1.3 DC Parametric Testing

This involves the checking of DC data sheet characteristics such as minimum/maximum output voltages versus output sink/source currents, fan out capability, power consumption, breakdown voltages and leakage currents. By its very nature, DC parametric testing involves measurements of voltages and

currents and hence does not constitute a challenge in terms of the two demands on test procedures mentioned in the beginning of this Chapter. Therefore this aspect of testing memories will not be discussed.

1.4 Location of Faults on a Memory Board

The problem of fault location on a memory board is of considerable practical importance. We are interested in locating faults to the memory chips, decoder chip, data registers or the data and address bussing structure. A fault model for a memory board taking into consideration faults associated with the memory chips, decoder logic, data registers, and bussing structure is presented in Chapter 4. The strategy of fault location is based on testing the functional units on a memory board in the following order: data registers, bussing structure, decoder chip, and finally memory chips. Thus, we first test the data registers; if they are found out to be fault-free we test the bussing structure, and so on. Finally, in order to test the memory chips we apply the functional and pattern sensitivity test procedures.

CHAPTER 2

FUNCTIONAL TESTING OF MEMORIES

2.1 Functional Test Procedures in Existence

Various functional test procedures known as "GALPAT," "Walking Ones," "Marching Ones," "Checkerboard," "GALTCOL" are used by industry [3,4,7]. The length of these test procedures are $O(n)$, $O(n^{3/2})$ or $O(n^2)$, where n is the number of cells in memory. These test procedures fail to satisfy the two demands on test procedures mentioned in the beginning of Chapter 1. The $O(n)$ tests which take a small amount of time have a limited fault coverage [5]; on the other hand, the $O(n^2)$ tests are reasonably powerful in fault coverage but consume a prohibitive amount of time, especially for 4K and 16K RAMs. The $O(n^{3/2})$ test procedures do not succeed in bridging the gap between these two extremes. Moreover, these test procedures seem to have been derived in a rather ad hoc fashion.

2.2 Basic Testing Philosophy for Memories

Our approach would be to formulate a fault model based on physical failures in the memory, and then derive a test set to detect the faults in the model. The fault model would not be on the "gate" level as in classical fault diagnosis [4] but would be formulated on a higher level in terms of functional blocks, like the decoder and the memory cell array.

Using such a fault model, a new test procedure for functional testing of memories is proposed. It depends on the successive division of memory. This procedure takes a time on the order of $n \cdot \log_2 n$ and is not only very powerful in its fault coverage ability but also achieves a drastic improvement in testing time over the $O(n^2)$ test procedures.

A schematic block diagram of the basic organization of a semiconductor RAM is shown in Figure 2.1. The actual details of the organization of a particular chip will depend on various factors like the technology used, the static or dynamic nature of the memory, the number of cells per word, and the actual location of memory cells and supporting logic on the chip. Usually the details of organization beyond the block diagram level and details of the chip layout are not available to test designers. Therefore the fault model, which is to provide a basis for the development of the functional test procedure, must be quite general in its nature and should not be restricted to peculiarities of a particular memory design.

Both functional testing as well as pattern sensitivity testing of a memory chip follow the basic procedure below:

Step 1: The tester writes a given test sequence, i.e., a sequence of test patterns into a given set of memory words.

Step 2: The tester reads a given set of memory words in sequence. As soon as the data from a word read is found out to be different from the expected data, the testing stops, as the memory is found out to be faulty. If the memory has passed the tests so far and there are more test patterns to be written, the tester returns to execute Step 1, otherwise it stops testing the memory which is declared functionally correct, or free of pattern sensitivity.

All test patterns are generated algorithmically. The tester's memory stores only the test program along with a few "initializing" test patterns. As will be seen later, the test program for detecting a faulty memory chip is quite small and hence does not require the tester to have a large memory.

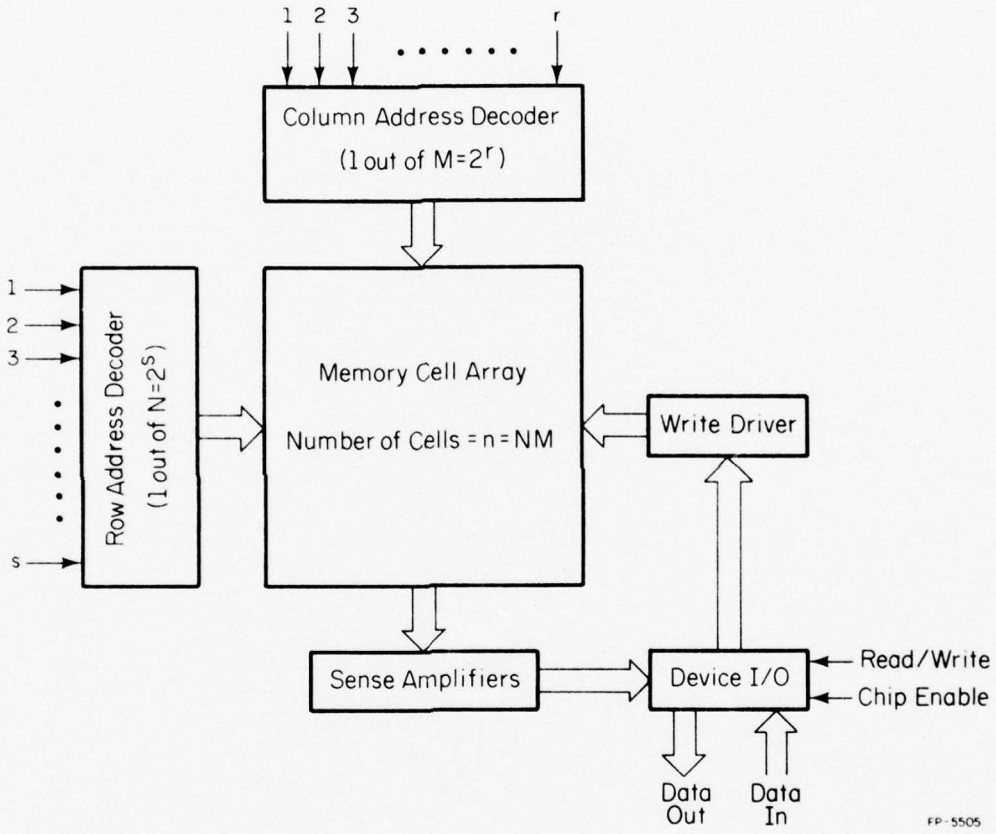


Figure 2.1. Block diagram of a semiconductor random access memory.

On the other hand, as described in Chapter 4, the test procedure for locating faults on a memory board is considerably more complicated due to the complex nature of the problem.

While writing and reading test patterns, the tester will have to follow various timing considerations associated with the memory under test, such as access time, data set-up and release time, address set-up and release time, various pulse widths and critical overlap conditions of different signals, and precharge and refresh requirements (if any).

2.3 Fault Model for Semiconductor RAM

As shown in Figure 2.1, a RAM can be divided into three functional blocks, i.e., memory cell array, decoder logic, and Read/Write logic (sense amplifiers, write drivers and other supporting logic).

2.3.1 Fault Model for Memory Cell Array

When the memory is accessed, the row decoder selects a particular row of cells (1 out of $N = 2^s$) and the column decoder selects a particular column of cells (1 out of $M = 2^r$), resulting in the access of the addressed word. Usually each memory word consists of a single cell. We will refer to this organization as an "nx1 organization," signifying that the memory has n words consisting of 1 cell each ($n = N \cdot M$).

In some memory designs like Motorola's MCM 6810 128x8-bit RAM or Intel's 8101-2 and 8111-2 256X4-bit RAMs, there is more than one cell per word, in which case a complete row of m cells will be accessed at one time. We will refer to this organization as an "nXm organization," signifying that it has n words consisting of m cells each, i.e., a total of nXm cells in the memory cell array.

Now let us consider the behavior of the memory cell array under failures associated with it. Consider a typical realization of a RAM cell shown in Figure 2.2.

In the following discussion positive logic is assumed. The discussion is valid for both p and n-channel realizations of a memory cell. Let us assume that with transistor T_1 on and T_2 off, the cell is storing a 1. If A is stuck at 0, the cell will fail to store a 0, as T_2 will never turn on. Similarly if B is stuck at 0, the cell will fail to store a 1. Similar arguments apply when A is stuck at 1 or B is stuck at 1. If the gate of T_1 or T_2 gets shorted to its drain, it will effectively short A to B, which in turn would form a wired-OR or a wired-AND function between the driving logic of A and B. Depending on the nature of this wired function (i.e., OR or AND) and the actual values of circuit parameters involved, the cell will either fail to store a 1 or a 0 or will fail to undergo a 0-1 or a 1-0 transition, i.e., a 0-1-0 transition. A break in the cross-coupling metallization will be "seen" as a stuck-at-1 or a stuck-at-0 fault by transistor T_1 or T_2 and the same discussion given above will apply.

Because of the extremely high density of cells on the chip, there may be shorts between two cells due to faulty metallization. Depending on whether the short is between transistors T_1 (or T_2) of two cells or between T_1 of one cell and T_2 of the other, these two cells will fail to store either the same logic values or the different logic values. In either case a transition in one cell will cause a transition in the other cell.

There may be capacitive coupling between two cells, say cells i and j. In this case a transition in one cell may change the contents of the other.

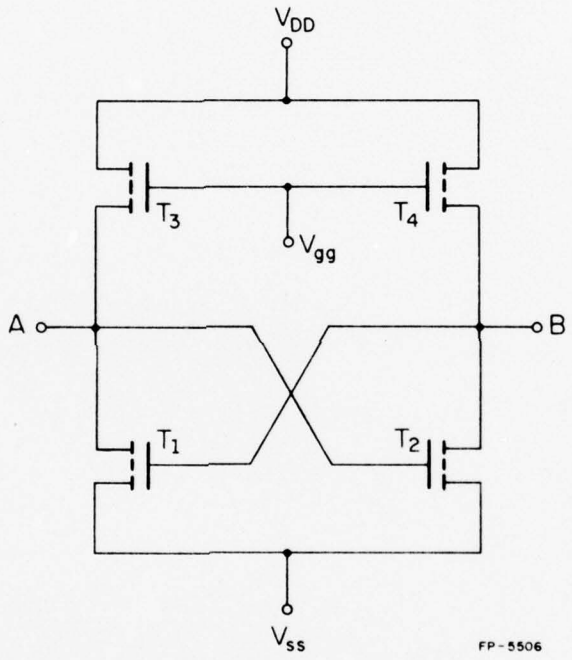


Figure 2.2. Typical realization of a RAM cell.

As shown in [6], the capacitive coupling behaves in a peculiar way, in that if a transition in cell i changes the state of cell j, then it does not necessarily imply that a transition in cell j will also change the state of cell i.

Based on this discussion the following fault model for the memory cell array is proposed. One or more of the following statements is true under failure:

1. One or more cells are stuck at 1 or 0.
2. One or more cells fail to undergo a 0-1-0 or a 1-0-1 transition.
3. There exist two (or more) cells, say cells i and j, which are coupled. By this we mean a 0 to 1 or a 1 to 0 transition in one of the two cells, say cell i, changes the state of the other cell, i.e., cell j. This does not imply that a similar transition in cell j will also change the state of cell i.

It may be pointed out that the O(n) test procedures are capable of detecting only those faults which are covered by statements 1 and 2 above. Though the fault model proposed for the memory cell array is based on the faulty behavior of a typical memory cell realization (Figure 2.2), it is characterized at a high level and it covers a wide variety of faults. It seems reasonable the model will be valid for different failure mechanisms in other realizations of a memory cell.

2.3.2 Fault Model for Decoder

The decoder is a simple combinational logic circuit that selects a unique memory cell for the given address. In other words, the decoder establishes a one-to-one correspondence between a given address and a memory cell. A block diagram of a 1-out-of-n decoder is shown in Figure 2.3. Now let us

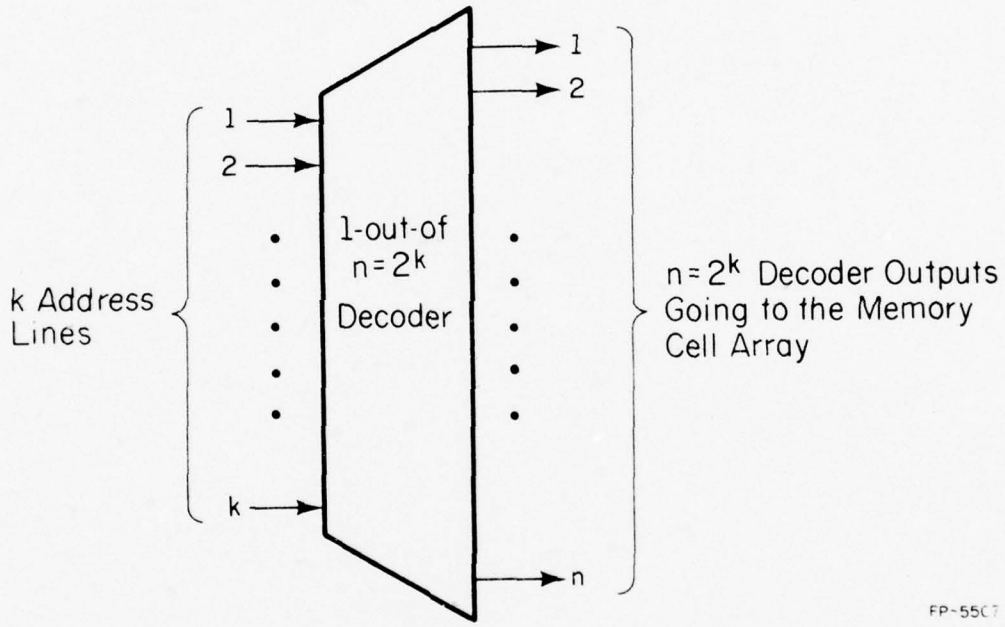


Figure 2.3. Block diagram of a 1-out-of-n decoder.

consider the behavior of the decoder under failures associated with it. Some output lines of the decoder may be permanently stuck at 1 or stuck at 0. This will result in the permanent selection of those cells that are accessed by the decoder output lines which are stuck at 1(0), and in the permanent non-selection of those cells that are "accessed" by the decoder output lines which are stuck at 0(1).

Some input line to the decoder (i.e., an address line) may be stuck at 1 or 0. This will result in the access of only half of the memory. In general, if k input lines are stuck, then only $\frac{1}{2^k}$ th portion of the memory can be accessed.

Some output lines of the decoder may be coupled together because of short circuits or capacitive coupling. Similar failures may occur with input (address) lines. Depending on whether this coupling results in a wired-OR or a wired-AND function, the decoder will access the non-addressed cells in addition to the addressed cell, or the decoder will fail to access the addressed cell.

The following fault model for the decoder is thus proposed. Any failure occurring in the decoder logic will make it behave in one of the following ways. For one or more address input(s), one or more of the following things can happen:

1. The decoder will not access the addressed cell. In addition, it may access non-addressed cells.
2. The decoder will access multiple cells, including the addressed cell.

In the case of multiple accesses, we can view the decoder fault as a memory cell array fault, i.e., as a coupling between memory cells. In the

case of no access, the cell which would have been selected under no fault, can be viewed as stuck at 1 or 0, depending upon the logic used. Thus we are, in effect, visualizing decoder faults as memory cell array faults. This is an important point in the development of the fault model for the entire semiconductor RAM, since we need not consider decoder faults explicitly.

2.3.3 Fault Model for Read/Write Logic

Some output lines of the sense amplifier logic or write driver logic (Figure 2.1) may be stuck at 1 or 0. In either case we can consider this fault as the same as stuck at 1 or 0 memory cells that correspond to the stuck output lines. The data input lines may have shorts or capacitive coupling between them. Similar may be the case with data output lines. This failure can be visualized as coupling between the memory cells that correspond to the coupled data input or output lines. We are, in effect, visualizing faults associated with the Read/Write logic as faults associated with the memory cell array.

Therefore, in order to test a semiconductor RAM, we need to concentrate on the fault model for the memory cell array only. Recalling the three statements regarding the fault model for the memory cell array presented at the end of Section 2.3.1, the functional test procedure must make the following transitions for every pair of memory cells, say for a pair consisting of cells i and j:

1. Change cell i from a 0 to a 1, once when cell j stores a 0 and a second time when it stores a 1.
2. Change cell i from a 1 to a 0, once when cell j stores a 0 and a second time when it stores a 1.

3. Repeat steps 1 and 2, after exchanging the roles of cells i and j.

It is easy to see that any test procedure which is capable of satisfying this requirement will be able to detect any cell which is stuck at 1 or 0. It will also detect any cell that fails to undergo a 0-1-0 or a 1-0-1 transition, as well as coupling between cells. Thus such a test procedure is able to detect all faults covered by the proposed fault model for a semiconductor RAM.

The development and presentation of such a functional test procedure is the topic of the next section.

2.4 Functional Test Procedure

2.4.1 Development of a Functional Test Procedure

In this section we first develop a functional test procedure for memory chips having an n×1 organization. As will be shown at the end of this section, the functional test procedure thus developed can be easily modified for memory chips having an n×m organization.

Let us imagine that a memory having an n×1 organization is partitioned into k equal parts, each containing $\frac{n}{k}$ cells, as shown in Figure 2.4. Assume that the whole memory is written with all 0's. We change all the cells of the first part from a 0 to a 1 and then read the cells of the remaining (k-1) parts. This step involves $\frac{n}{k}$ WRITE operations and $\frac{k-1}{k} \cdot n$ READ operations, i.e., a total of n operations, where n is the number of cells in the memory. The above step is repeated for a 1-0 transition in all the cells of the first part. Finally the two steps performed so far, a 1-0 and a 0-1 transition in all the cells of the first part, are repeated when complementary data are present in the remaining (k-1) parts (in this example the (k-1) parts contain all 1's).

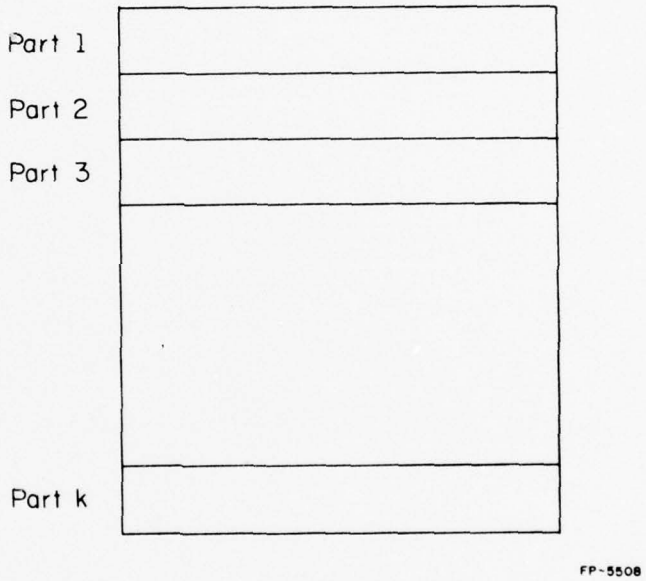


Figure 2.4. Memory partitioned into k equal parts.

Thus we have made a 0-1 and a 1-0 transition in the first part, while the remaining (k-1) parts are in a given state as well as its complementary state. Recalling the fault model for memory presented in Section 2.3, particularly the four transitions mentioned at the end of Section 2.3.3, we can see that the operations performed so far ensure that a transition in any cell of the first part does not change the state of any cell in the remaining (k-1) parts. We must repeat these operations for the remaining (k-1) parts. This will ensure that there is no coupling between cells of any two different parts.

As we have seen earlier, a transition in any part requires $\frac{n}{k}$ WRITE operations, and in order to read the remaining (k-1) parts $\frac{k-1}{k}.n$ READ operations are needed. These operations are to be performed for both transitions, i.e., a 0-1 and 1-0 transition when the remaining (k-1) parts are in a given state as well as in the complementary state. This will require a total of $4(\frac{n}{k} + \frac{k-1}{k}\ n) = 4n$ operations. Since these operations are to be repeated for each of the k parts, at least 4nk operations will be required to ensure that there is no coupling between cells of any two different parts among these k parts.

It is important to note that with any arbitrary sequence of transitions in the k parts, it will not be possible to complete the test with 4nk operations, since many transitions would be redundant. The minimum of 4nk operations will be achieved only if each part undergoes a given transition (i.e., a 0-1 or a 1-0) only two times, once for the given state and once for the complementary state of the remaining (k-1) parts.

Assuming that the memory is written with all 0's in the beginning, a typical set of these 4nk operations is given in Figure 2.5, in a matrix form. Each row of the matrix corresponds to a part of the memory, i.e., the i^{th} row corresponds to the i^{th} part. Each column of the matrix corresponds to a transition. An up arrow, denoted by ↑, indicates a 0-1 transition, while a down arrow, denoted by ↓, stands for a 1-0 transition. Since each column of this matrix corresponds to n operations, $\frac{n}{k}$ WRITE operations + $\frac{k-1}{k}$.n READ operations, the test matrix comprises 4nk operations.

The 0-1 transition in a typical part, say the i^{th} part (corresponding to the i^{th} row in Figure 2.5) is performed only two times, the first time when the state of the remaining (k-1) parts is given by the $(i+1)^{st}$ column of the test matrix and the second time when the state of the remaining (k-1) parts is given by the $(2k+i+1)^{st}$ column of the test matrix, as shown in Figure 2.5. Thus for the two times when a 0-1 transition takes place in the i^{th} part the states of the remaining (k-1) parts are complementary to each other. Similarly the 1-0 transition in the i^{th} part is performed only two times, and the states of the remaining (k-1) parts are complementary to each other at the time of these two transitions. A careful study of Figure 2.5 will show that each part of the memory undergoes a 0-1 and a 1-0 transition twice, and the states of the remaining (k-1) parts are complementary to each other at the time of these transitions. Therefore this test matrix represents the minimum number of operations (4nk operations).

However, this test matrix is not unique. It is not difficult to develop an algorithm to generate all possible test patterns in terms of a k-cube. The k-cube has 2^k vertices and $k \cdot 2^{k-1}$ edges. Each edge corresponds

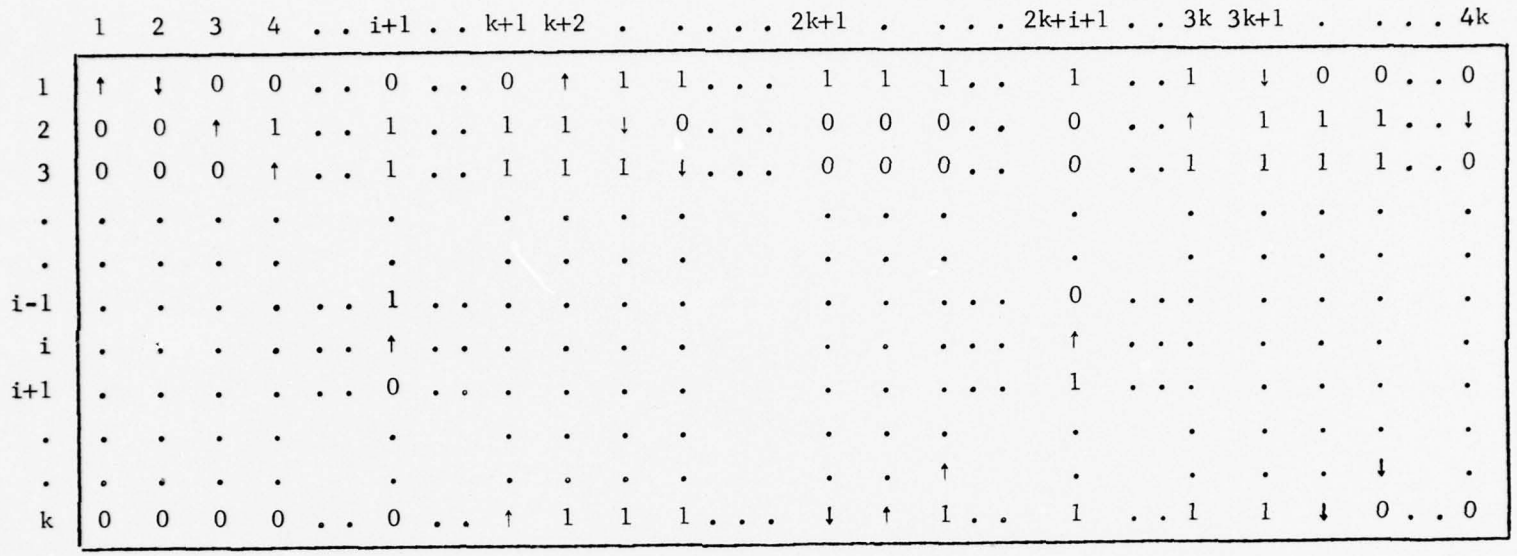


Figure 2.5. A typical test matrix showing the minimum 4nk operations.

to a transition in one of the k parts. If each edge is replaced by two directed edges, in opposite directions, corresponding to the two transitions (1-0 and 0-1), any algorithm that generates all possible traversals of the edges of the (modified) k-cube, subjected to the following conditions will be able to give all possible test patterns.

1. Start traversing edges from the (0 0...0) vertex of the k-cube.
2. Traverse 4k directed edges in such a way that no edge is traversed twice and 2k of the traversals correspond to the 2k transitions with a given set of states, while the remaining 2k traversals correspond to the complementary 2k transitions.
3. At the end of the $4k^{th}$ traversal, it must be possible to return to the origin, i.e., to the (0 0...0) vertex.

Since the intention here is to show the existence of test patterns consisting of 4nk operations for any value of k (as demonstrated by the test pattern in Figure 2.5), we do not give an exact algorithm to this effect.

Now each of these k parts obtained previously is partitioned into k equal parts and the same sequence of tests is run on the $k \times k = k^2$ parts. With (4nk + 4nk) = 8nk operations, we can ensure that these k^2 parts do not have any coupling between them. We can repeat the same procedure for the parts obtained by partitioning each part of the previous run into k parts, until we come to a stage where each part corresponds to a single cell in the memory cell array. Let us assume that with p iterations we arrive at this stage. Then with 4nkp operations we can ensure that there is no coupling between any two arbitrary cells of the memory because this set of operations satisfies the four requirements of transitions for every pair of cells, given

at the end of Section 2.3.3. Since every cell is made to undergo a 0-1 and a 1-0 transition, it is also checked whether any cell is stuck at 0 or 1, or is unable to undergo a 0-1-0 transition. Thus all the test requirements described in terms of the fault model presented in Section 2.3 are satisfied.

Variables k and p are related by

$k^p = n$ = number of cells in the memory.

$$\therefore p = \frac{\ell n(n)}{\ell n(k)} .$$

Let T_n = the total number of operations performed

$$= 4nkp = 4n \cdot \ell n\ n \cdot \frac{k}{\ell nk}$$

It is easy to verify that k = e gives the minimum number of operations. Since we cannot partition a memory into e parts the nearest integer to e, i.e., 2 or 3 must be chosen. The choice of 3 will need a slightly fewer number of operations than the choice of 2, but it is not a natural choice as all available memories consist of a number of cells which is a power of 2. Therefore the memory is to be partitioned into two equal parts in successive runs. With k = 2,

T_n = the total number of operations performed

$$= 8n \cdot \log_2 n.$$

The functional test procedure developed for the n×1 organization of the memory can be easily adapted to the n×m organization as explained below. The memory with an n×m organization is partitioned into two equal parts in successive runs of the functional test procedure, until we come to a stage where each part corresponds to a single word of m cells in the memory cell

array. The number of operations performed so far is $8n \cdot \log_2 n$. With this set of operations we can ensure that there is no coupling between the cells of any two different words of the n words of the memory cell array. We define this set of operations as word-wise partitioning.

In order to ensure that there is no coupling between the cells of the same word, we must perform similar partitioning operations on the cells of memory words and apply a similar test pattern. We define this set of partitioning operations performed on the cells of memory word as cell-wise partitioning. This set of operations involves $16n \cdot \log_2 m$ operations. The factor 16 appears instead of 8 because we cannot read only half of a word while performing the cell-wise partitioning. The total number of operations for testing a memory with an nXm organization = $8n \cdot \log_2 n + 16n \cdot \log_2 m = 8n \cdot \log_2 nm^2$. (Note that the total number of cells in the memory = $n \cdot m$).

2.4.2 Functional Test Patterns

One approach to derive a functional test pattern is to modify the test matrix of Figure 2.5 for k = 2. Another approach that (in addition) yields all possible test patterns for k = 2, is to employ a "binary test-tree" and is presented here for the nXl memory organization.

Let us assume that to begin with the memory under test is written with all 0's. Right in the beginning we have two choices of transitions, i.e., we can change either the top half of the memory from a 0 to a 1 or the bottom half from a 0 to a 1. Similarly at each successive run we have a choice of changing the top half or the bottom half. Our aim is to make a 0-1 as well as a 1-0 transition in one half when the other half contains first all 0's and then all 1's, with the minimum possible operations, i.e., 8n operations.

A systematic enumeration process is needed to keep track of various transitions. This can best be done with the help of a binary tree, whose top branch corresponds to a transition in the top half of the memory and whose bottom branch corresponds to a transition in the bottom half of the memory. (We can conveniently use the terminology left half and right half of the memory when describing the cell-wise partitioning used for testing a memory with an n×m organization.)

One of the four possible test patterns which ensures that the two halves of a memory are uncoupled is derived in Figure 2.6 with the help of a binary tree. Figure 2.7(a) shows this test pattern in a tabular form. By completing the remaining tree in the same fashion it is possible to obtain three more test patterns. These three additional test patterns are shown in Figure 2.7(b), (c) and (d). These are the only possible test patterns that utilize the minimum number of operations.

Any one of these test patterns can be employed for the functional testing of an n×1 memory organization, as well as for the cell-wise partitioning operations performed while testing an n×m memory organization.

2.5 Algorithms for the Functional Test Procedure

In this section algorithms for the functional test procedure for n×1 and n×m memory organizations are presented. It is important to note that the operations performed in the functional testing of an n×1 memory organization are the same as those performed in the word-wise partitioning for an n×m memory organization, because a word of memory with an n×1 organization consists of a single cell. Therefore the algorithm WORD-WISE PARTITIONING presented in Section 2.5.1 for the functional testing of an n×1 memory organization can

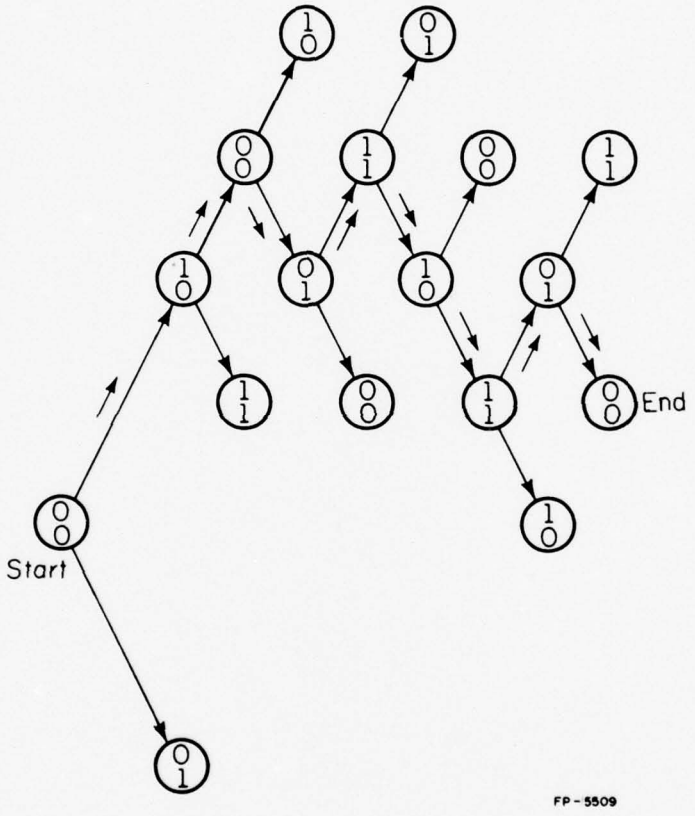
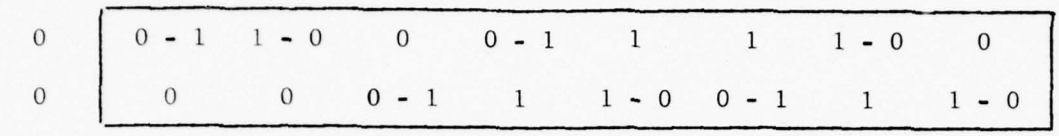
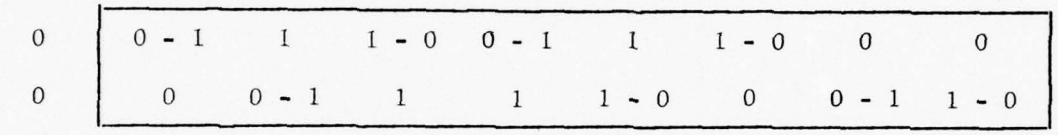


Figure 2.6. Derivation of a test pattern using a binary tree.

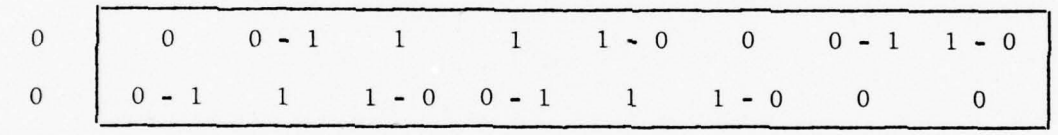
START



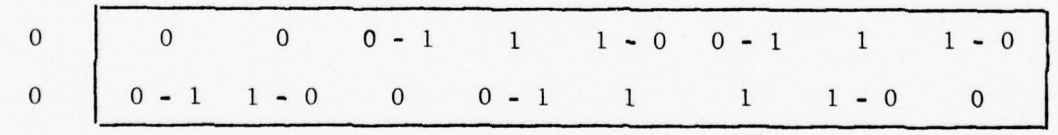
(a)



(b)



(c)



(d)

Figure 2.7. Four possible test patterns.

perform the word-wise partitioning while testing an nXm memory organization. Algorithm CELL-WISE PARTITIONING presented in Section 2.5.2 performs the cell-wise partitioning while testing an nXm memory organization.

2.5.1 Development of Algorithm WORD-WISE PARTITIONING

Two procedures, WRITEMEM and READMEM, are used in this algorithm. As their names suggest, procedure WRITEMEM writes specified memory words with the specified data and READMEM reads specified memory words expecting the specified data, and transfers control to an error location if the expected data is not read, indicating a fault in the memory chip under test.

Both the procedures and algorithm WORD-WISE PARTITIONING are written in an ALGOL-like high level language. Control signals given to the tester-memory interface hardware are written in English.

As explained in Section 2.4, we partition the memory while performing the functional testing. For a memory with an nXl organization, $k = \log_2 n$ partitioning runs are needed. During the i^{th} partitioning run ($1 \leq i \leq k$), the memory is divided into 2^i blocks, each containing 2^{k-i} memory words. Exactly half the blocks (2^{i-1} blocks) are written with either a 0 or 1, and remaining 2^{i-1} blocks are read. SIZE is a global variable designating the number of words in a block.

Procedure WRITEMEM has two parameters, d and A; d is the specified data to be written in all the words of the 2^{i-1} blocks (during the i^{th} partitioning run), and A specifies the starting address of the first block belonging to the set of 2^{i-1} blocks. Procedure WRITEMEM follows.

```
PROCEDURE WRITEMEM (d,A);
BEGIN
    X ← A; BLOCKS ← 1;          /* number of blocks written into */
    WHILE BLOCKS ≤ 2^(I-1) DO   /* I the parameter for partitioning run */
          BEGIN
              FOR J ← 1 TO SIZE DO
              Write data d in memory word with address X + J - 1;
              X ← X + 2 * SIZE;
              BLOCKS ← BLOCKS + 1
          END
END;
```

Procedure READMEM has two parameters, d and A, where d is the expected data to be read in all the words of the 2^{i-1} blocks (during the i^{th} partitioning run), and A specifies the starting address of the first block belonging to the set of 2^{i-1} blocks.

Procedure READMEM is given below.

```
PROCEDURE READMEM (d,A);
BEGIN
    X ← A;
    BLOCKS ← 1;
    WHILE BLOCKS ≤ 2^(I-1) DO
          BEGIN
              FOR j ← 1 TO SIZE DO
              Read memory word with address X + J -1; if the data read is not
              d, flag error and exit.
```

```
            X ← X + 2 * SIZE;
            BLOCKS ← BLOCKS + 1
        END
END;
```

With these procedures algorithm WORD-WISE PARTITIONING follows.

Algorithm WORD-WISE PARTITIONING:

```
BEGIN
    INITIALIZE:  Write the entire memory with 0's;
    K ← log2n;
    FOR I ← 1 TO K DO
    BEGIN
        SIZE ← 2^(K-I)
        WRITEMEM (1,0)    ;  READMEM (0,SIZE);
        WRITEMEM (0,0)    ;  READMEM (0,SIZE);
        WRITEMEM (1,SIZE);   READMEM (0,0)    ;
        WRITEMEM (1,0)    ;  READMEM (1,SIZE);
        WRITEMEM (0,SIZE);   READMEM (1,0)    ;
        WRITEMEM (1,SIZE);   READMEM (1,0)    ;
        WRITEMEM (0,0)    ;  READMEM (1,SIZE);
        WRITEMEM (0,SIZE);   READMEM (0,0)    ;
    END
END;
```

It may be noted that this algorithm uses the functional test pattern given in Figure 2.7(a). Any other test pattern given in Figure 2.7(b), (c) or (d) is equally good. We shall adhere to the test pattern of Figure 2.7(a) in algorithm CELL-WISE PARTITIONING too.

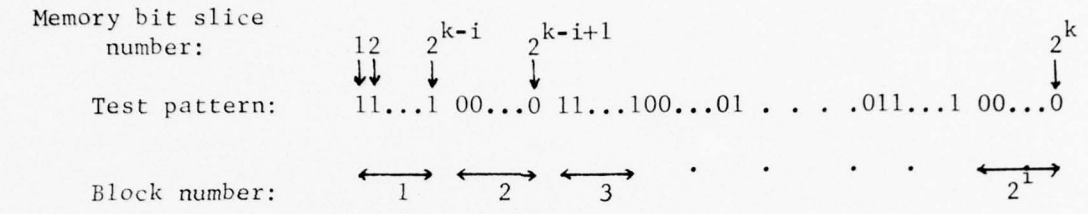
2.5.2 Development of Algorithm CELL-WISE PARTITIONING

This algorithm performs the cell-wise partitioning in the functional testing of an nXm memory organization. A memory with an nXm organization is said to have m bit slices, such that the i^{th} bit slice is the set of i^{th} cells of all the words in memory (labelling the most significant cell of a word as the 1st cell and the least significant cell of a word as the m^{th} cell).

As explained in Section 2.4, while performing the cell-wise partitioning, we partition the memory so that each partition or block contains specified bit slices. For a memory with an nXm organization, $k = \log_2 m$ cell-wise partitioning runs are needed. During the i^{th} partitioning run ($1 \leq i \leq k$), the memory is divided into 2^i blocks, each containing 2^{k-i} bit slices. Exactly half the blocks (2^{i-1} blocks) are written with 0(1), and remaining 2^{i-1} blocks are written with 1(0). During every partitioning run the entire memory is read.

For the complete cell-wise partitioning, a total of $2 \cdot \log_2 m$ test patterns is used. Since m is usually a small number (8 or 16), all these $2 \cdot \log_2 m$ test patterns can be stored in the tester memory and can be retrieved using a table look-up. These test patterns are given below.

The test pattern used during the i^{th} cell-wise partitioning run is given in Figure 2.8(a). This pattern is numbered pattern I. Its bit-wise complementary pattern will also be used during the i^{th} cell-wise partitioning run. The bit-wise complementary pattern of pattern I is denoted by $\overline{\text{pattern I}}$. Patterns 1, 2 and 3 to be used for an nX8 memory organization are shown in Figure 2.8(b). Their bit-wise complementary patterns will also be used. With these test patterns algorithm CELL-WISE PARTITIONING follows.



Each block contains 2^{k-i} bit slices.

(a) Pattern I

Pattern I: IIII 0000

Pattern 2: 1100 1100

Pattern 3: 1010 1010

(b) Three patterns for an n×8 memory organization.

Figure 2.8

Algorithm CELL-WISE PARTITIONING:

```
BEGIN
    INITIALIZE:  Write the entire memory with 0's;
    K ← log2m;
    FOR I ← 1 TO K DO
    BEGIN
          Write the memory with pattern I; Read the memory;*
          Write the memory with all 0's  ; Read the memory;
          Write the memory with pattern I (complemented); Read the memory;
          Write the memory with all 1's  ; Read the memory;
          Write the memory with pattern I; Read the memory;
          Write the memory with all 1's  ; Read the memory;
          Write the memory with pattern I (complemented); Read the memory;
          Write the memory with all 0's  ; Read the memory;
    END
END;
```

*While reading the memory, if the data read from any word does not conform to the pattern just written, an error is flagged and program execution is stopped as a fault is detected in the memory chip.

CHAPTER 3

PATTERN SENSITIVITY TESTING

Pattern sensitivity testing is an important aspect of the overall testing of semiconductor memories. Under certain combinations of data patterns and sequences of memory accesses, various device anomalies and parasitic effects can manifest themselves in pattern sensitivity problems because of the following:

1. Change in the dynamic timing parameters of the memory, and
2. Peculiarities of the layout and design of the memory chip.

As mentioned in Section 1.2, our discussion will be limited to the pattern sensitivity problem caused by changes in the dynamic timing parameters. In order to check completely for pattern sensitivity, one has to use every possible sequence of memory accesses with every possible data pattern. This will obviously take a prohibitive amount of time. A practical approach to this problem is to build a "worst case" model of pattern sensitivity and develop test patterns to generate these worst case conditions so that if any pattern sensitivity is present, it will be detected. Such a model is developed here in terms of various delays and dynamic timing parameters of memory and then test procedures are presented in terms of these parameters.

A note of caution is needed here. The model for pattern sensitivity proposed in this Chapter may not be able to account for every possible kind of pattern sensitivity associated with the dynamic timing parameters of every existing type of memory. This is due to the fact that considerable variation exists among the dynamic timing parameters and their interrelationships for

different types of memories, particularly between static and dynamic memories. Therefore it is very difficult to construct a model for pattern sensitivity applicable to every existing type of memory. However, the model proposed here can be applied to many types of memories and the basic ideas can be used to generate tests for pattern sensitivity for other types.

3.1 Pattern Sensitivity Model

All dynamic timing parameters of memory can be described in terms of delays associated with various signal paths in memory. These delays depend on data and address and herein lies the root of pattern sensitivity. The memory delay model along with important dynamic timing parameters is described below [8].

As shown in Figure 3.1, a memory is modelled as an "ideal" memory with four types of input and one type of output. By definition, the ideal memory has no intrinsic delays, but all its external signal paths have delays associated with them. These delays are defined below.

ΔI: The length of time required to propagate a change in the input data at the input data pins to the ideal memory.

ΔO: The length of time required to propagate a change in the output data from the ideal memory to the output data pins.

ΔA: The length of time required to propagate a change in the address from the address pins to the ideal memory.

ΔW: The length of time required to propagate a change in the signal at the WRITE ENABLE pin (also known as Read/Write pin, as shown in Figure 2.1) to the ideal memory.

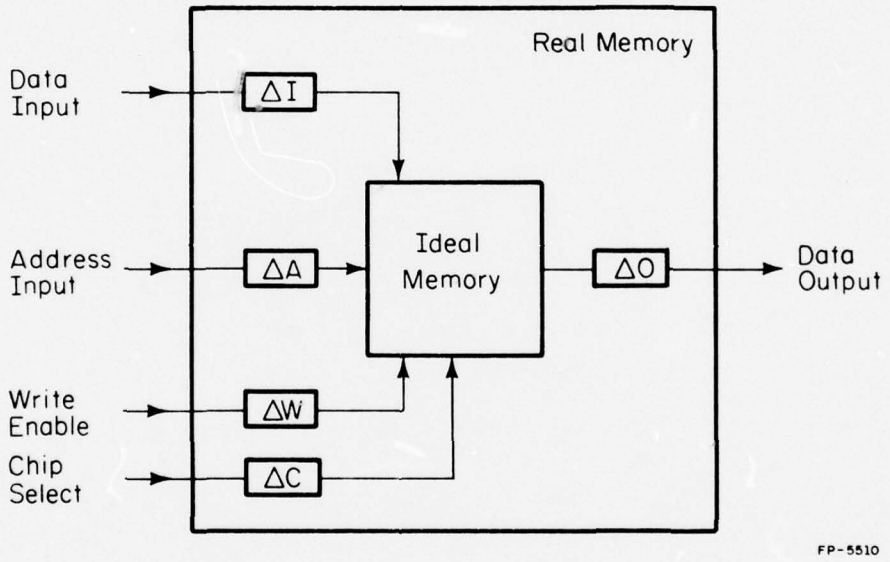


Figure 3.1. Memory delay model.

ΔC: The length of time required to propagate a change in the signal at the CHIP SELECT pin (also known as Chip enable pin, as shown in Figure 2.1) to the ideal memory.

The data input path is either a single or a multi-line data bus. Each line in this bus is associated with a delay ΔI and the values of ΔI associated with two different lines need not be the same. The same is true for the data output bus and ΔO delay. ΔW and ΔC are the delays associated with the single signal paths. ΔA is quite complex in its nature, as the path associated with ΔA includes the decoder, decoder latches, and buffers. Therefore it is not possible to associate ΔA with individual address pins or address paths. Instead, ΔA is associated with an address change. Two different address changes need not have the same value of ΔA. Moreover, each input and output shown in Figure 3.1 is binary in nature. Therefore each ΔI, ΔW, ΔC, ΔO and ΔA will have two different values, one for the propagation of a 1-0 transition and the other for the propagation of a 0-1 transition.

As shown in [8], all dynamic timing parameters of the memory can be treated as some combination of these delays. Since these delays depend on the data and address transitions, the values of the dynamic timing parameters also depend on the data and address transitions, giving rise to pattern sensitivity. There are eight important dynamic timing parameters associated with memory. They are described below.

1. Access time (t_{acc}): The length of time from the appearance of a valid address on the address pins to the appearance of valid data on the data output pins in a READ operation. The maximum value of t_{acc}, $t_{acc_{max}}$, specifies how long the system must wait to get the valid data output after it supplies the address.

2. Read Recovery time (t_{rr}): This specifies the time period for the output of a memory to return to its quiescent state, after the address has been removed. It is an important parameter when two successive READ operations are to be performed.

3. Write Enable time (t_w): The length of time for which the active level must be present on the WRITE ENABLE pin to guarantee a successful WRITE operation.

4. Data Set-Up and Release time: The maximum value of data set-up time ($t_{ds_{max}}$) and the minimum value of the data release time ($t_{dr_{min}}$) are of significance. The new data to be written must be stable at the data input pins no later than $t_{ds_{max}}$ before the WRITE ENABLE pulse ends at the WRITE ENABLE pin, and this data should remain stable at least for $t_{dr_{min}}$ from the end of the WRITE ENABLE pulse.

5. Address Set-Up and Release time: The maximum value of the address set-up time ($t_{as_{max}}$) and the minimum value of the address release time ($t_{ar_{min}}$) are of importance. The address of the word to be written must be stable at the address pins no later than $t_{as_{max}}$ before the WRITE ENABLE pulse is applied. The address should change no earlier than $t_{ar_{min}}$ before the end of the WRITE ENABLE pulse.

6. Write Recovery time (t_{wr}): In many memory designs, the sense amplifiers respond to a WRITE operation. t_{wr} specifies the time taken by the data output pins to return to the quiescent state after a WRITE operation. The use of the data output pins is prohibited for t_{wr} units after a WRITE operation. This parameter is of significance when a READ operation follows a WRITE operation.

7. Chip Select delays: These are described by two parameters; the chip enable time (t_e) gives the time period between the leading edge of the chip select pulse and obtaining the output in the active state, and the chip disable time (t_d) gives the time period between the trailing edge of the chip select pulse and the restoration of memory output to the passive state. These parameters are of importance in a multi-chip operation.
8. Sense Amplifier recovery: If one tries to read a long series of cells containing the same data, followed by a cell containing the complementary data, then a sense amplifier may fail to register the change in data due to its excessively large recovery time.

On the basis of this memory delay model, which relates the dynamic timing parameters to delays associated with various signal paths in memory, we describe the worst case conditions to generate the extreme values of the dynamic timing parameters. This will detect pattern sensitivity related to changes in the dynamic timing parameters. The generation of these worst case conditions depends on the following fact about the decoder logic [6]. An address change involving a change in the logic level of every address pin is the slowest to propagate through the decoder; on the other hand, an address change involving a change in the logic level of a single address pin is the fastest to propagate through the decoder.

On the basis of this fact, we propose test procedures (Section 3.2) for detecting pattern sensitivity related to each dynamic timing parameter. As might be noticed all these test procedures are $O(n)$ or $O(n \cdot \log_2 n)$.

3.2 Pattern Sensitivity Test Procedures

In this section we propose test procedures for detecting pattern sensitivity associated with access time, address set-up and release time, data set-up and release time, and write recovery time. It is critical that the minimum or maximum values of the dynamic timing parameters as suggested by the test procedures in this section be adhered to.

3.2.1 Access Time Sensitivity

A change in an address involving a change in the logic level at every address pin creates the maximum value of the access time parameter, t_{acc}. Therefore if the access time corresponding to every such address transition is less than the maximum access time specified for the memory, the memory will not suffer from the access time sensitivity problem. A test procedure for detecting excessive value of the access time parameter (t_{acc}) is described below.

We write the upper half of memory, i.e., cells 1 through $\frac{n}{2}$, with all 0's, and the lower half, i.e., cells $(\frac{n}{2}+1)$ through n, with all 1's. We then read cell 1, followed by cell n, and finally cell 1 again (i.e., read address 000..0, address 111..1, address 000..0); this is followed by the sequence cell 2, cell (n-1), cell 2; ...; cell $\frac{n}{2}$, cell $(\frac{n}{2}+1)$, cell $\frac{n}{2}$. This pattern is repeated with the complementary data, i.e., the upper half of memory written with all 1's and the lower half with all 0's. Thus we are making every address transition involving a change in the logic level at all address pins. Any address transition corresponding to excessive access time will be detected, since in that case the data read (within the access time interval) will be

opposite to the data expected. This procedure involves 2n WRITE operations and $(2x3x\frac{n}{2})$ = 3n READ operations, i.e., a total of 5n operations.

3.2.2 Address Set-Up and Release Time Sensitivity

An address change involving a transition in the logic level of every address pin causes the maximum value of the address set-up time. On the other hand an address change involving a change in the logic level of a single address pin causes the minimum value of the address release time.

A test procedure for checking excessive address set-up time follows. We write cell 1 with a 1; then cell n is written with a 0 and cell 1 is read. This pattern is repeated for the cell 2 - cell (n-1) pair, ..., cell $\frac{n}{2}$ - cell $(\frac{n}{2}+1)$ pair. Then the pattern is repeated for the sequence of cell n - cell 1 pair, cell (n-1) - cell 2 pair, ..., cell $(\frac{n}{2}+1)$ - cell $\frac{n}{2}$ pair. Finally the entire pattern described so far is run for the complementary data. This procedure involves a total of $2x2x3x\frac{n}{2}$ = 6n operations. If any address transition has excessive address set-up time, it will change the contents of the first cell of a pair of cells when the second cell of the pair is being written. This will be detected when the first cell is read after the second cell is written.

For detecting excessively low address release time the procedure given below can be followed. The whole memory is written with a background pattern of all 0's. A 1 is written in cell 1 and an address change is made by changing the logic level of a single address pin, $t_{ar_{min}}$ time period before the WRITE ENABLE pulse ends at the WRITE ENABLE pin. The cell thus addressed is read. If a 0 is not read, the presence of excessively low address release time will be detected. It is easy to see that we have to make k such address

transitions for every test cell, where $2^k = n$. The pattern is to be repeated for the complementary data. The procedure involves $2(n + n + kn) = 4n + 2n \cdot \log_2 n$ WRITE operations and $2kn = 2n \cdot \log_2 n$ READ operations, i.e., a total of $4n + 4n \cdot \log_2 n$ operations.

3.2.3 Data Set-Up and Release Time Sensitivity

Here we are interested in detecting excessively high data set-up time and excessively low data release time. For this we write the first cell with the data input pins undergoing a 0-1 transition at $t_{ds_{max}}$ time period before the WRITE ENABLE pulse ends at the WRITE ENABLE pin, followed by a 1-0 transition at $t_{dr_{min}}$ time period before the WRITE ENABLE pulse ends at the WRITE ENABLE pin. This pattern is repeated for every cell. Then the whole memory is read. Finally the entire pattern is repeated for the complementary data. Any cell that suffers from an excessively high data set-up time or an excessively low data release time will be written to the opposite of the intended value. This will be detected when that cell is read. The total number of operations is 4n, consisting of 2n READ and 2n WRITE operations.

3.2.4 Write Recovery Time Sensitivity

Here we want to detect the presence of excessively high WRITE RECOVERY time, t_{wr}. Suppose the quiescent logic state of the data output pins is a logic 1(0). We write a cell, say cell A, with a 1(0). Then we write a cell, say cell B, whose address is complementary to that of cell A, with a 0(1) and read cell A. If t_{wr} is excessively large so that it covers the maximum access time specified for the memory, the data read from cell A will not be a 1(0). Thus, we write cell 1 with the quiescent logic state of the data output pins, write cell n with the opposite data and read cell 1.

This operation is repeated in the reverse order, i.e., we write cell n, write cell 1 and read cell n. The sequence so far is repeated for the cell 2 - cell (n-1) pair, ..., cell $\frac{n}{2}$ - cell $(\frac{n}{2}+1)$ pair. The total number of operations is 3n.

Pattern sensitivity problems can also be caused by slow sense amplifier recovery. A suitable test pattern called "SHIFTED DIAGONAL" is described in [3]. As mentioned earlier, in addition to pattern sensitivity caused by variations in dynamic timing parameters, in practice pattern sensitivity arising from peculiarities of the design and layout of a memory chip must be taken into consideration. Case studies regarding pattern sensitivity of some existing memory chips like Intel's 1103 memory and Motorola's MCM 6605 memory are given in [9] and [10], respectively.

Dynamic semiconductor memories must also be tested for their capability to hold their information accurately for a specified period of time. This is known as refresh testing. This problem is discussed in [10], [11], and [12].

CHAPTER 4

LOCATION OF FAULTS ON A MEMORY BOARD

Faults on a memory board can be readily detected by applying the algorithms derived for testing a memory chip to the memory board as a whole. In this Chapter we discuss the problem of locating faults on a memory board and give an approach for solving it.

Figure 4.1 shows an example of a memory board housing:

1. An MxK array of nx1 semiconductor memory chips, comprising n·K words of memory of M bits each. A chip placed in the i^{th} row and j^{th} column is denoted by chip (i,j).
2. 1-out-of-K decoder logic to select one out of the K columns of memory chips.
3. An address bus consisting of $\log_2 nK$ lines, such that $\log_2 K$ high order lines feed the 1-out-of-K decoder logic and $\log_2 n$ low order lines are used to address each memory chip.
4. Input and output data registers each M bits wide.
5. Input and output data busses each M bits wide.

4.1 Fault Model for a Memory Board

We will use the following fault model for a memory board.

1. The fault model for memory chips and decoder logic is the same as that described in Section 2.3.
2. Since the input and output data registers consist of M cells each, the fault model for these data registers is the same as that for a single memory word of M cells. Thus, under a fault, one or more cells of the data

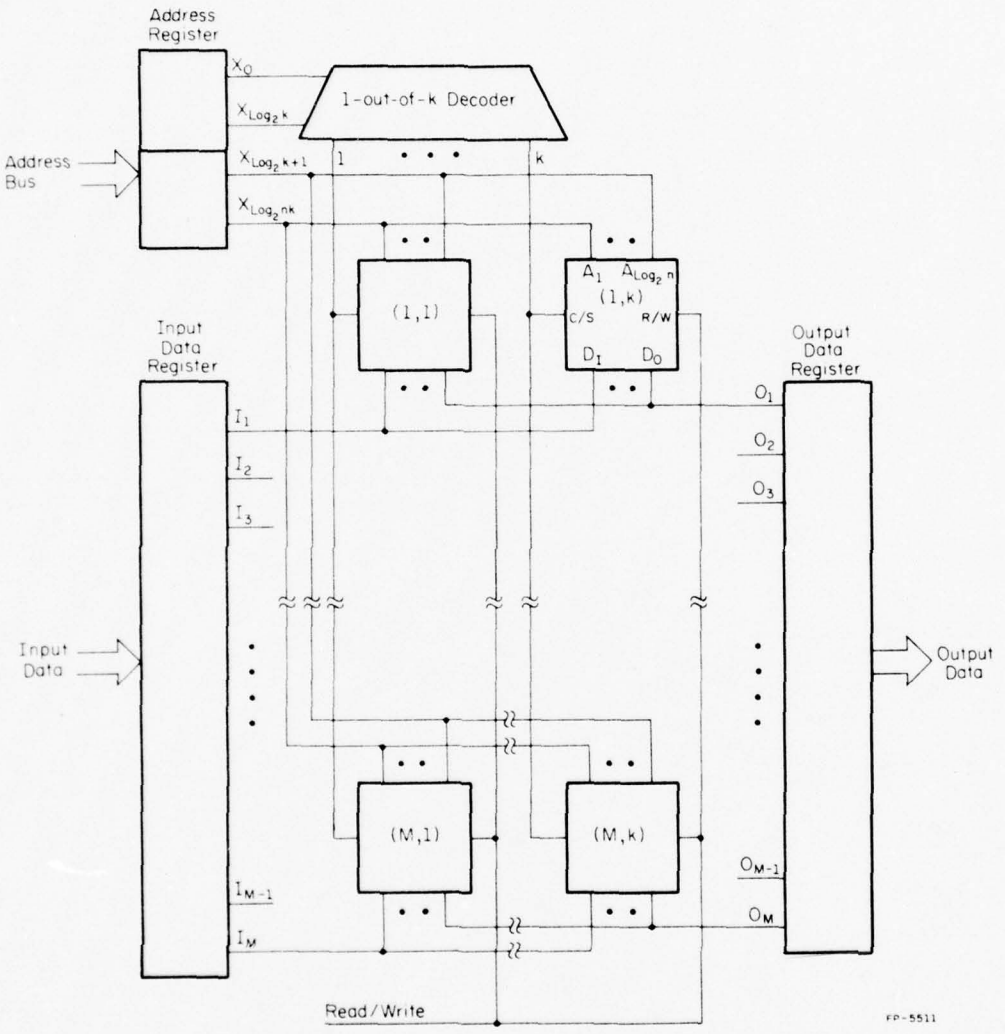


Figure 4.1. Organization of a memory board.

register are stuck at 0 or 1, or fail to undergo a 0-1-0 transition. In addition two or more cells may have coupling between them.

3. Any line belonging to a bus may be stuck at 0 or 1, or there may be coupling between two or more lines of a bus due to metallization shorts or inductive or capacitive linkage. Since only the $\log_2 n$ lower order address lines constitute the effective address bus on the board, we assume that coupling could also exist between lines of the data busses and these lower order lines of the address bus.

The location procedure will be derived under the assumption that there are no breaks in the bus lines. The motivation for such an assumption is twofold. It seems reasonable to assume that the probability of a break occurring during regular operation is very low. In addition, if such an assumption is not made, the fault location problem becomes much more complex because of interaction with other faults. As an example, if the input data bus line 1 breaks just after chip (1,2), we will be able to read from and write into the first two chips in the first row, i.e., chip (1,1) and (1,2) but cannot write into any of the other chips in the row. Thus only part of the bus is affected and such a fault can interfere with the effects of other faults (a decoder fault, for example). Hence we will assume that under a fault the entire bus line is stuck at a constant value or is coupled to other bus lines.

4. It is important to note that two cells belonging to two different memory chips cannot have any coupling between them.

4.2 Rationale Behind the Fault Location Test Procedure

We will use the following notation for a memory address. An address A applied to access a word can be split into two parts, A_i and a_j, where A_i

corresponds to the set of high order $\log_2 K$ address lines (which selects a column out of the K columns of memory chips), and a_j corresponds to the set of low order $\log_2 n$ address lines (which accesses a cell in the memory chips belonging to the selected column). Thus $A_1 a_1$ corresponds to address 00...0 00...0, and $A_K a_n$ corresponds to address 11...1 11...1.

It is assumed that the tester can access the memory board only through the input data port, output data port, and address port. Under these conditions many fault equivalence classes exist. For example, if the k^{th} line of the output data bus is stuck at 1(0), it cannot be distinguished from the k^{th} cell of the output data register stuck at 1(0). In fact, the k^{th} cell of the output data register stuck at 1(0), the k^{th} line of the output data bus stuck at 1(0), outputs of all memory chips in the k^{th} row stuck at 1(0), the k^{th} line of the input data bus stuck at 1(0), and the k^{th} cell of the input data register stuck at 1(0) form a fault equivalence class. As another example, a fault that causes a non-selection of the i^{th} column when address $A_i a_j$ is applied is equivalent to a fault that causes outputs of all memory chips in the i^{th} column stuck at 1 or 0 (depending on the logic used).

It is a very tedious and time consuming task to find out all fault equivalence classes present in this system. A better approach would be to locate a fault to a fault equivalence class on reading faulty output data, using the knowledge of the past history of test outputs (if necessary). One way to attempt to "distinguish" the faults within a fault equivalence class is to run a suitable test sequence and depending on the test output data of this sequence, decide which fault has the maximum chance of occurrence. Thus the purpose of such a test sequence is to increase the confidence level in

isolating the faults belonging to a fault equivalence class. Such tests in our discussion will be called confidence tests.

4.3 Development of the Fault Location Test Procedure

We break down this task into the development of test procedures for,

1. Checking that no lines of data busses and no cells of data registers are stuck at 1 or 0.
2. Checking that no coupling exists between the input and output data busses, or between the address bus and the data busses.
3. Checking that no coupling exists between lines within a data bus, or between cells of a data register.
4. Checking that the decoder is fault-free.
5. Checking that every memory chip is fault-free. Otherwise we want to locate every faulty memory chip.

1, 2, and 3 above verify that the bussing structure and data registers are fault-free. These test procedures (1 through 5) will be applied in sequence to the memory board under test. Flowcharts are used to aid in the explanation of the test procedures. The index i used in the flowcharts designates the i^{th} column of memory chips on the memory board.

Using any word with address $A_i a_1$ ($1 \leq i \leq K$) as the test word, we check whether it is possible to read a 0 on every output data line (not necessarily simultaneously), by writing the test word with data 00...0 and reading it. If this is possible, we can conclude that no line of a data bus, and no cell of a data register is stuck at 1. The test word with address $A_i a_1$ is chosen to mask the effects of any coupling between a data bus and the address bus (as lines of all busses are held at 0). If it is not possible to read a

0 on a line of the output data bus with any test word $A_i a_1$, we write a reasonably large number of words with data 00...0 and read them. This test sequence is called confidence test 1A.

If any output data line remains at logic 1 throughout this test, then with a high degree of confidence we can conclude that either the data registers have a stuck-at-1 cell, or the data busses have a stuck-at-1 line (these two faults form a fault equivalence class). At this stage we may replace the data registers and run the test procedure from the beginning, if we have not replaced the data registers earlier. If we have replaced them before, we can conclude that the data busses have a stuck-at-1 line.

If no line of the data busses or no cell of the data registers is stuck at 1, we can run a similar test using any word with address $A_i a_n$ ($1 \leq i \leq K$) as the test word and input data 11...1 to check if any line of the data busses or any cell of the data registers is stuck at 0. Confidence test 1B can be defined analogously. These two tests are given in the flowcharts in Figures 4.2(a) and (b), respectively. If a memory board passes these two tests, we can conclude that no lines of the data busses and no cells of the data registers are stuck at 1 or 0.

We now proceed to check if any coupling exists between the input and output data busses, or between a data bus and the address bus. We check whether it is possible to store 00...0 in the word with address $A_i a_1$, and 11...1 in the word with address $A_i a_n$, for some some i ($1 \leq i \leq K$). If this is not possible, we can immediately conclude with a high degree of confidence that the decoder is faulty (consider the flowchart in Figure 4.3). If the above test succeeds (say for i = I), then using the test word with address

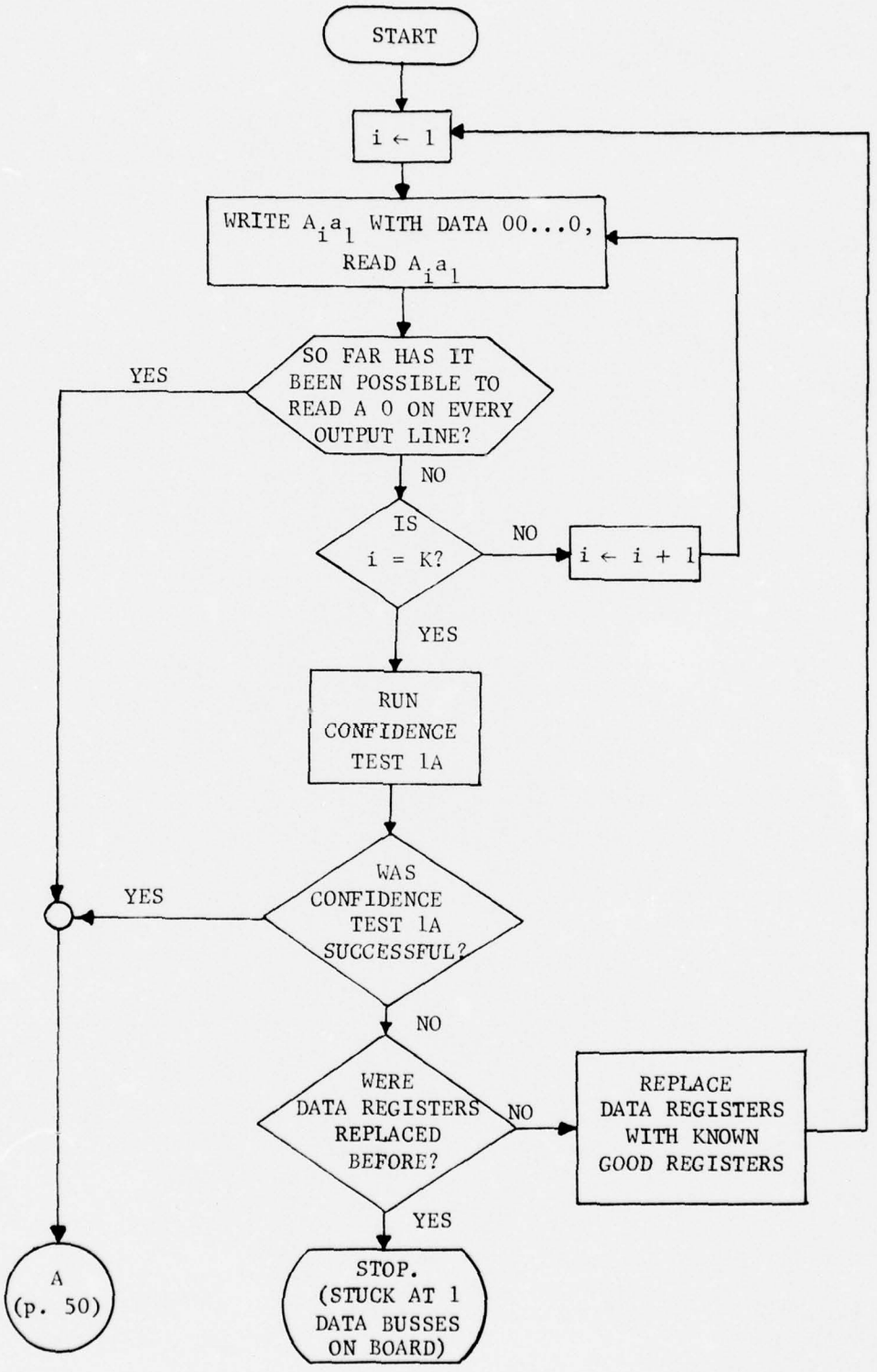


Figure 4.2(a). Test procedure for testing data busses.

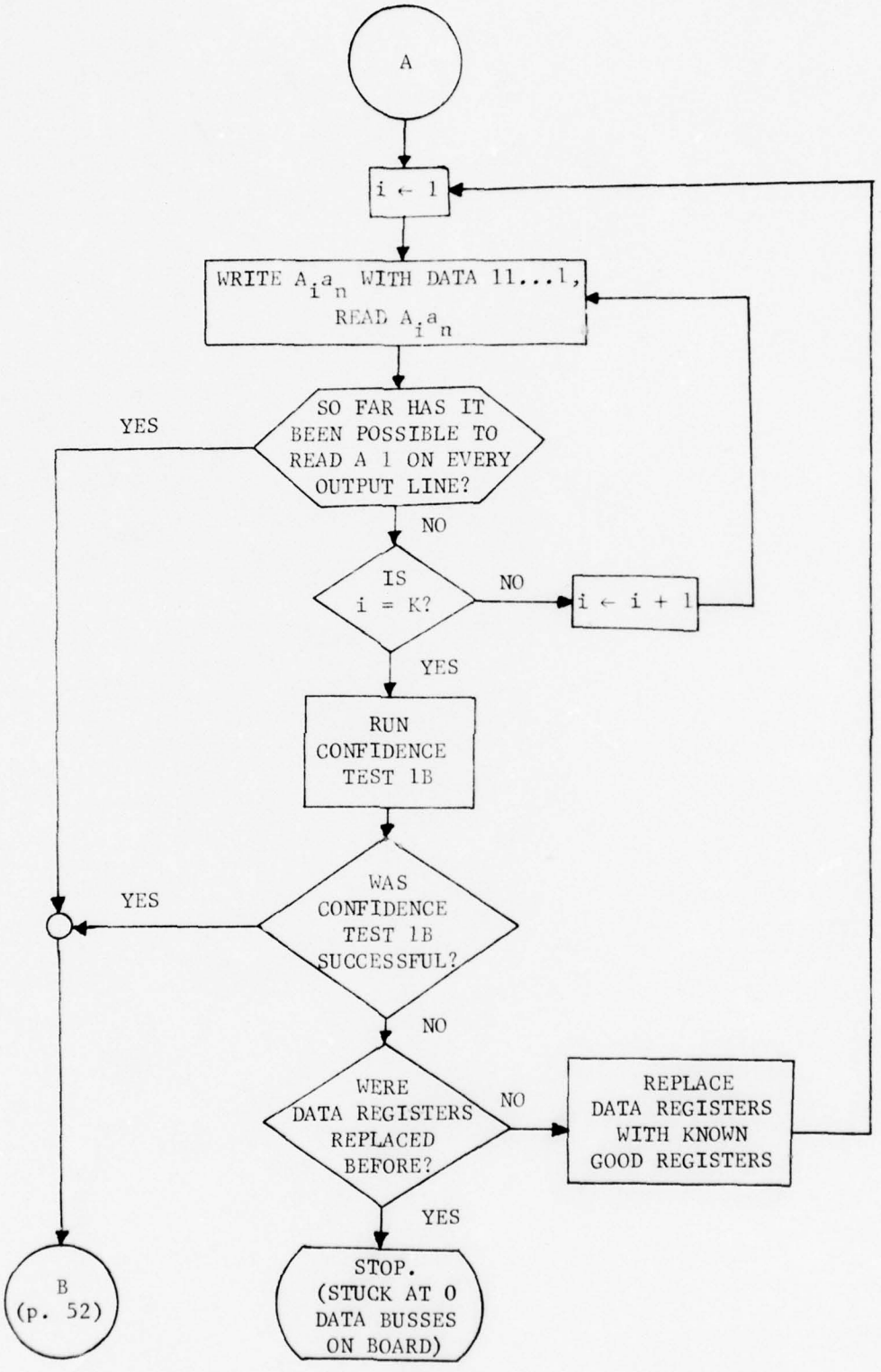


Figure 4.2(b). Test procedure for testing data busses.

$A_I a_1$ we write data 11...1, reset the input data bus to 00...0 and read the test word. If the output data is not 11....1, the address bus may be coupled to the data bus, or coupling exists between the input and output data busses, or the corresponding memory cell or memory chip output is stuck at 0.

In order to isolate faults within this class, we write many other words having addresses $A_i a_1$ ($1 \leq i \leq K$) with data 11...1, reset the input data bus to 00...0, and read those test words. If there is a coupling between the address and data busses, or between the input and output data busses, we will not be able to read 11..1 at any time during this test. This test is called confidence test 2A. If we fail to read 11..1 throughout this test, we conclude that a coupling exists between the different busses. If we are able to read 11..1 from a test word with address $A_I a_1$, then using a test word with address $A_I a_n$ we write 00..0, reset the input data bus to 11...1 and read the test word. If we are able to read 00...0, we conclude that no coupling exists between different busses.

If we cannot read 00...0 from the test word, we run a similar test in which we write many other words having addresses $A_i a_n$($1 \leq i \leq K$) with data 00...0, reset the input data bus to 11...1, and read those words. This test is called confidence test 2B. These tests are given in the flowcharts in Figure 4.3. At this stage we can conclude that there is no coupling between the data and address busses.

From the tests performed above we know that it is possible to store 00..0 in the word with address $A_I a_1$ and 11...1 in the word with address $A_I a_n$. We now check if it is possible to store 11..1 in $A_I a_1$, and 00...0 in $A_I a_n$. If it is not possible to do so, we replace the memory chips corresponding to the

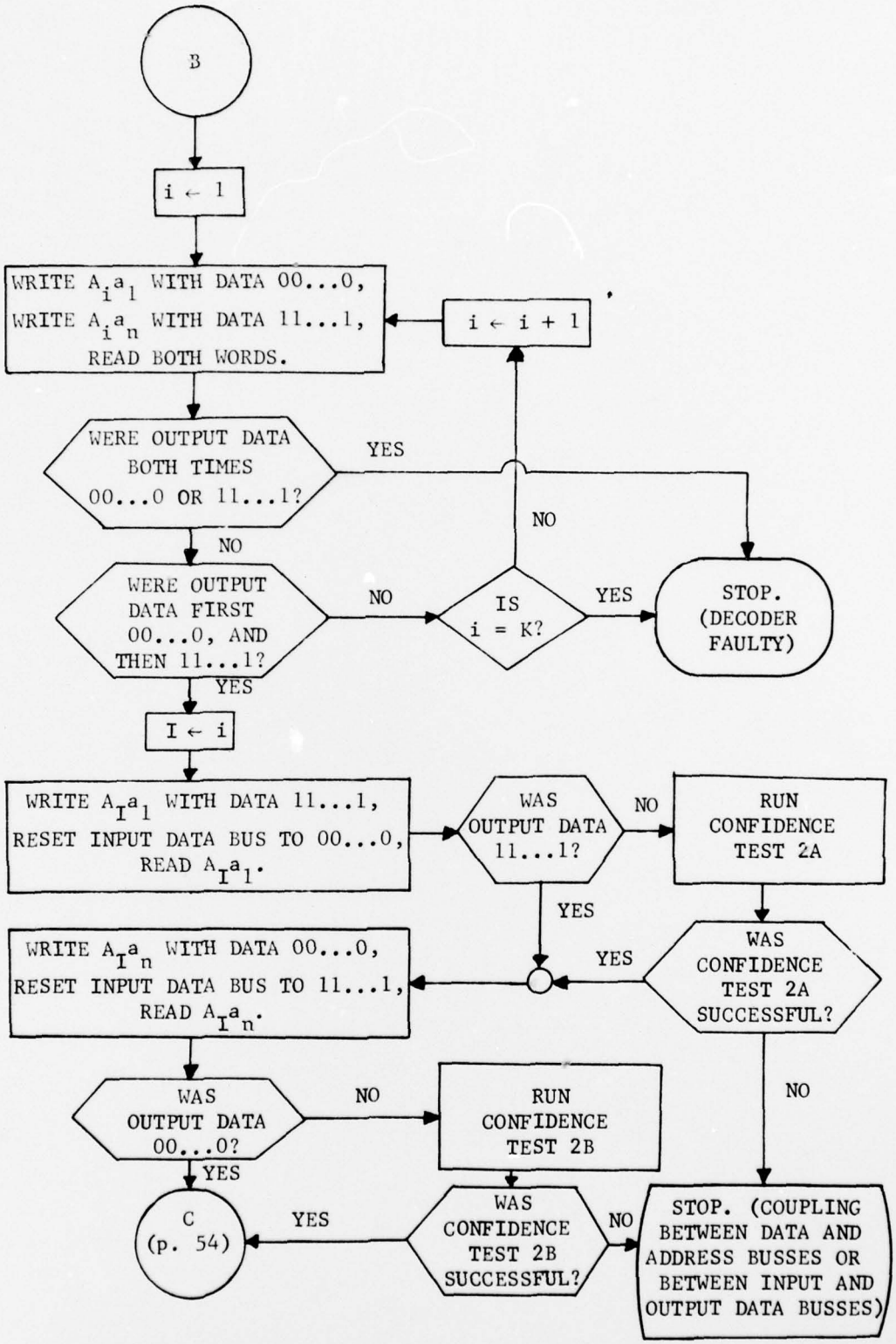


Figure 4.3. Test procedure for checking coupling between different busses.

rows giving a faulty output in the I^{th} column, as we know that these chips are faulty. If it is possible to store 11..1 in $A_I a_1$ and 00..0 in $A_I a_n$, we run the functional test procedure using the test patterns and the algorithm CELL-WISE PARTITIONING given in Section 2.5.2, and considering the word with address $A_I a_1$ or $A_I a_n$ as a memory (of course, comprising a single word) in its own right. If this test succeeds, there is no coupling between the lines of the input data bus, or between the lines of the output data bus, or between the cells of the data registers.

If this test fails, we conclude that the lines of the data busses, or the cells of the data registers are coupled, since no coupling can exist between cells of two different memory chips. Thus we replace both data registers and run this test again, using the same word, i.e., either $A_I a_1$ or $A_I a_n$. If the test fails again, we conclude that the lines of a data bus are coupled (we cannot distinguish whether the coupled lines belong to the input or the output data bus). The flowchart for this test is given in Figure 4.4.

At this stage we know that only the decoder, memory chips, and address bus need to be tested. We proceed to test the decoder first. We check if it is possible to find one word in every column of memory chips that can store both 00...0 and 11...1. We denote such a word in the i^{th} column by address $A_i a^i$ (a^i, in general, has no relation with a_i). On the other hand, suppose we fail to store 00..0 and 11...1 in any word in the j^{th} column, we can conclude that the decoder is faulty since the probability of having output of all memory chips in this column stuck at 1 or 0 is very low indeed; but if we find that only on certain rows we get a consistent 0 or 1 output, then the memory chip corresponding to that row and the i^{th} column is definitely faulty. We

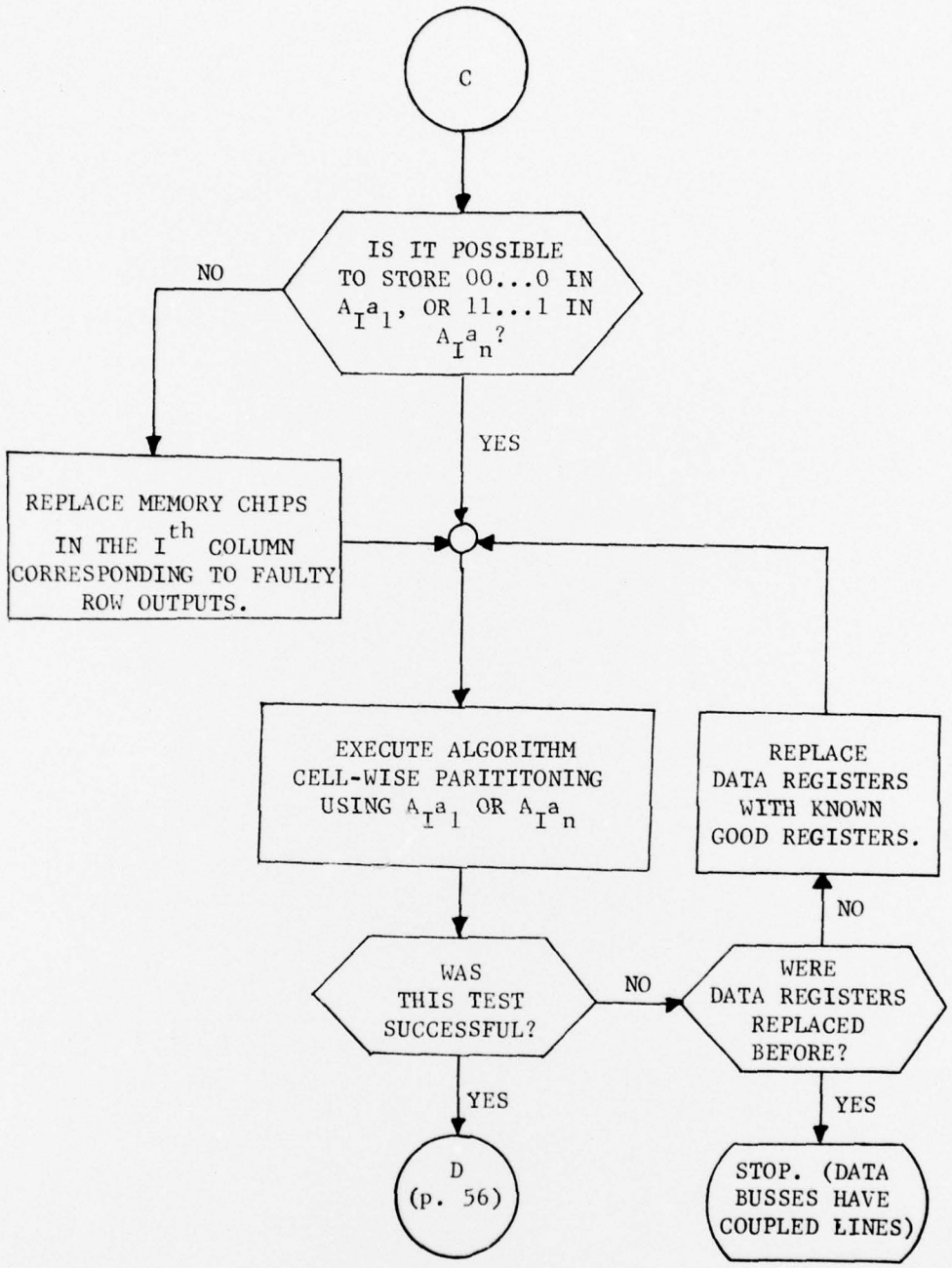


Figure 4.4. Test procedure for checking coupling between lines of a data bus.

replace such chips and proceed. Thus at this stage it is possible to find a word with address $A_i a^i$ in each column that can store 00...0 and 11...1.

In order to check the decoder, we apply the word-wise partitioning procedure, using the algorithm WORD-WISE PARTITIONING given in Section 2.5.1 on the set of words with addresses $A_i a^i$. If this test succeeds we can conclude that the decoder is fault-free; on the other hand, if the test fails, we conclude that the decoder is faulty since there can be no coupling between cells of different memory chips. The flowchart given in Figure 4.5 shows this test.

Finally, we apply the functional test procedure using the algorithm WORD-WISE PARTITIONING given in Section 2.5.1 for all the memory chips in a column simultaneously, and to successive columns sequentially. If at any stage erroneous output data is read, either the memory chips in the corresponding rows in the column being tested must be faulty or the address bus must be faulty (having stuck or coupled lines). If the cause is a faulty address bus, it should show the same faulty behavior for all memory columns (for example, half of the words in each column are stuck at 1 or 0, if an address line is stuck). Otherwise, with the help of the record of faulty output data we can locate the faulty memory chips as those chips lying in the corresponding rows and in the column being tested. The flowchart given in Figure 4.6 shows this test. Pattern sensitivity tests can also be run on all the memory chips in a column simultaneously, and on the successive columns sequentially.

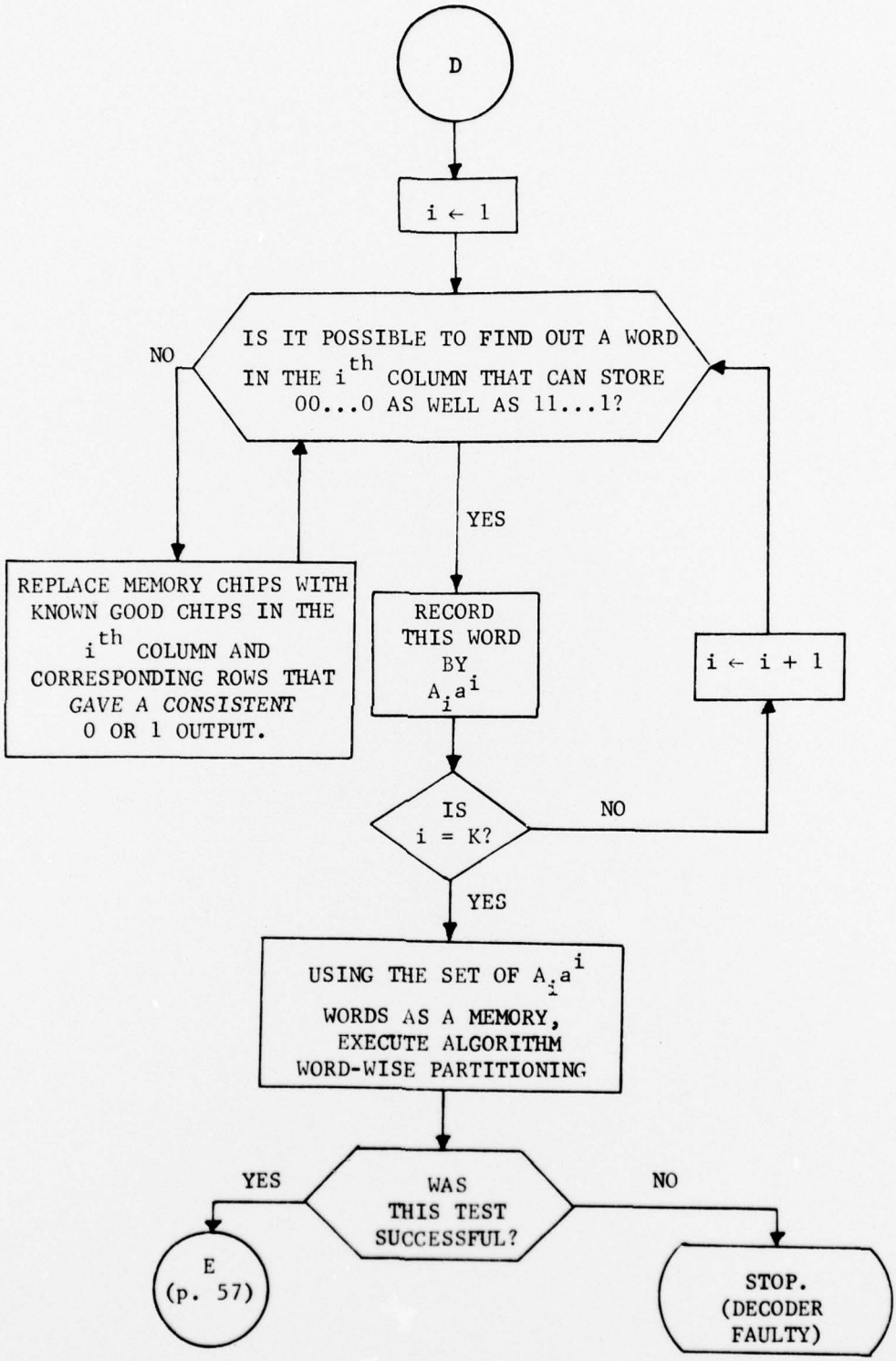


Figure 4.5. Test procedure for testing decoder.

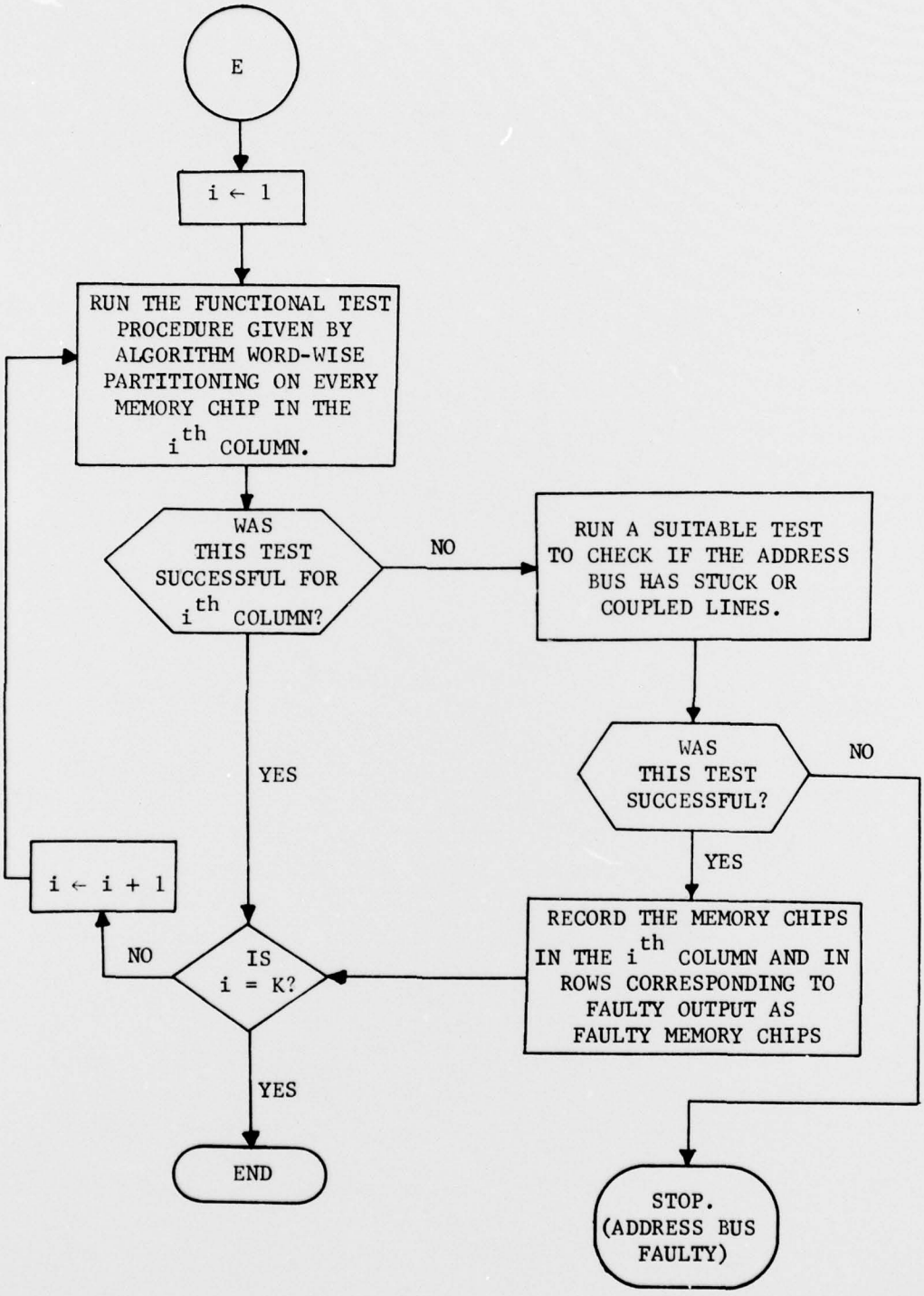


Figure 4.6. Test procedure for testing memory chips and the address bus.

CHAPTER 5

EVALUATION OF TEST PROCEDURES

In this Chapter we evaluate the functional test procedure, the pattern sensitivity test procedure, and the test procedure proposed for locating faults on a memory board. Three criteria for evaluating a test procedure for semiconductor memories are:

1. The fault coverage, i.e., "what does it find?"
2. The total time needed, and
3. The simplicity in computing addresses of words to be accessed, input data in a WRITE operation, expected output data in a READ operation, and READ and WRITE signals. These factors influence the complexity of the test generation algorithm and the cost of the tester.

5.1 Functional Test Procedure

As described in Sections 2.3.1 and 2.4.1, the functional test procedure is able to detect any memory cell which is stuck at 0 or 1, or which fails to undergo a 0-1-0 transition. In addition, the functional test procedure is able to detect a coupling between any two arbitrary cells in the memory cell array. It also detects any decoder faults covered by the decoder fault model described in Section 2.3.2 and any Read/Write logic fault covered by the fault model described in Section 2.3.3. On this basis we claim that the functional test procedure has a very good fault coverage ability.

Now let us consider the total time needed for the operations performed in the functional test procedure. Recalling from Section 2.4.1, for a memory having an $n \times 1$ organization, the total number of operations performed

is given by $8n \cdot \log_2 n$. Assuming that each operation (READ or WRITE) takes 500 nsec, the testing times for performing the functional test on 1Kx1, 4Kx1, and 16Kx1 RAMs are given in Table 5.1. The table also shows the corresponding time taken by the GALPAT procedure which requires $4n^2$ operations. The substantial decrease in testing time using the proposed functional test procedure is evident from Table 5.1.

As described in Section 2.4.1, for a memory having an nxm organization, the total number of operations performed is given by $8n \cdot \log(nm^2)$. Again assuming that each operation takes 500 nsec, the proposed functional test procedure needs only 64 msec for testing a 1Kx8 RAM. On the other hand the GALPAT procedure would require 128 seconds for testing the same RAM.

Recently some $0(n^{3/2})$ test procedures have been proposed that are more comprehensive in their fault coverage than the simple 0(n) procedures, and take a lesser amount of time than the $0(n^2)$ procedures. These $0(n^{3/2})$ test procedures are not as comprehensive in the fault coverage as the $0(n \cdot \log_2 n)$ procedure, and moreover the $0(n \cdot \log_2 n)$ procedures will always require less testing time than the $0(n^{3/2})$ procedures for example, the "GALTCOL" procedure [7], which is an $0(n^{3/2})$ procedure, checks for coupling between cells in the same column of the memory cell array and takes 760 msec (with a 470 nsec cycle time) for a 4Kx1 RAM, while the proposed functional test procedure takes only 180 msec and checks for coupling between any two arbitrary memory cells.

Regarding the third evaluation criterion for a test procedure, the proposed functional test procedure is quite simple as it is based on successive partitioning of the memory cell array and yields easy algorithms as seen in

Table 5.1

Comparison of time required for testing

Type of Test Procedure	Number of Operations	Time		
		1Kx1 RAM	4Kx1 RAM	16Kx1 RAM
Proposed Functional Test Procedure	$8n \cdot \log_2 n$	40 ms	192 ms	896 ms
GALPAT	$4n^2$	2 s	32 s	512 s

Section 2.5. However, as far as this criterion is concerned, $O(n)$ and $O(n^2)$ test procedures also yield easy algorithms [4].

Two approaches can be taken for implementing the functional test procedure depending on the memory space - execution time trade-off. The first approach is to generate all the address sequences used in the functional test procedure and store them in the tester memory. This would need memory space proportional to $n \cdot \log_2 n$, where n is the number of words in the memory under test. These address sequences can be retrieved by a table look-up when the test patterns are to be written. Obviously this approach sacrifices memory space for reducing the execution time.

In the second approach, an address of a memory word is generated only when the memory is to be accessed. This approach would not require any memory space for storing address sequences; but at the cost of increased execution time. Algorithm WORD-WISE PARTITIONING follows the second approach, as the time required for the $O(n \cdot \log_2 n)$ procedure is substantially lower than that required for an $O(n^2)$ procedure allowing us the option for a substantially smaller tester memory in the memory space - execution time trade-off. On the other hand, while performing the cell-wise partitioning we always write and read the entire memory in every partitioning run and there are very few test patterns ($\log_2 m$) to be written. Therefore in Algorithm CELL-WISE PARTITIONING we store the test patterns in the tester memory and retrieve them using a table look-up when writing them in the memory under test.

5.2 Pattern Sensitivity Test Procedure

As pointed out in the beginning of Chapter 3, the model for pattern sensitivity is based only on the variations of dynamic timing parameters with

change in data patterns or sequence of memory accesses. Therefore the pattern sensitivity test procedures proposed in Chapter 3 should be treated as a basic framework for testing pattern sensitivity. Additional tests need to be developed for a particular memory depending on how its dynamic timing characteristics differ from those covered in Chapter 3. Moreover the design and layout of the memory chip must be carefully studied in order to develop tests to identify pattern sensitivity related to idiosyncrasies of the design and layout of the memory chip.

Since all the procedures for detecting pattern sensitivity are $O(n)$ or $O(n \cdot \log_2 n)$, they will take a small amount of time. In addition, as presented in Section 3.2, these test procedures are quite simple and use very easy algorithms in terms of the third evaluation criterion.

5.3 Fault Location Test Procedure for a Memory Board

The problem of locating faults on a memory board is of considerable practical importance. It was shown that the problem is quite complex in its nature due to the interaction between faults in the decoder logic, bussing structure, and memory chips. A fault location test procedure is given. Even though the test procedure is quite complex, it requires only about 5 seconds for locating faults on a 64Kx16 bit memory board (housing 64 16Kx1-bit RAMs), since the length of the test procedure is $O(K \cdot n \cdot \log_2 n)$.

CHAPTER 6

CONCLUDING REMARKS AND SUMMARY

The problems of testing semiconductor random access memories and of locating faults on a memory board housing memory chips, decoder logic, data registers, and bussing structure were discussed. Memory testing was divided into functional testing and pattern sensitivity testing. Fault models for memory failures as well as pattern sensitivity phenomenon were developed. Based on these models algorithms for functional testing and pattern sensitivity testing were presented. It was shown that the problem of fault location on a memory board is a very difficult problem because of a large number of fault equivalence classes. The strategy of using confidence tests was applied to break the fault equivalence classes in order to locate the faults.

Finally test procedures for functional testing, pattern sensitivity testing, and fault location testing were evaluated applying the criteria of fault coverage, time requirement and ease of implementation. It was shown that the functional test procedures have a very good fault coverage, provide a dramatic improvement in testing time over the $0(n^2)$ test procedures, and are easy to implement.

It is very difficult to construct a comprehensive model for pattern sensitivity as the phenomenon is very closely related to the device parameters, peculiarities of design and layout of the memory chip, and various parasitic and anomalous effects. Therefore the pattern sensitivity test procedures should be treated as a basic framework; additional tests may be needed depending on the peculiarities of the memory chip under test.

The functional test procedure has been successfully implemented in the Microcomputer Laboratory at the University of Illinois. The test program was written in the National Semiconductor IMP-16 assembly language to test a 4Kx16 bit memory board housing 64 Intel 2125 (1Kx1 bit) chips. On detection of a fault on the board, the user can interactively run the test to try to locate the fault. Faulty chips are easily found. On one occasion, the test procedure aided in the location of an address bus to data bus short on the board. The test procedure is being implemented on other systems at the University of Illinois. We feel that the functional test procedure is ideal for testing single memory chips as well as for periodically testing memory modules on a computer system.

REFERENCES

1. Hayes, J. P., "Detection of Pattern-Sensitive Faults in Random Access Memories," IEEE Trans. on Computers, Vol. C-24, pp. 150-157, February 1975.

2. Colbourne, E. D., Coverley, G. P., and Behra, S. K., "Reliability of MOS LSI Circuits," Proc. IEEE, Vol. 62, pp. 244-259, February 1974.

3. Fischer, J. E., "Test Problems and Solutions for 4K RAMs," in Dig. Semiconductor Test Symposium, Cherry Hill, NJ, IEEE Computer Society, pp. 53-71, November 1974.

4. Breuer, M. A. and Friedman, A. D., "Diagnosis and Reliable Design of Digital Systems," Computer Science Press, Inc., 1976, pp. 139-145.

5. Knaizuk, J. and Hartman, C. R., "An Optimal Algorithm for Testing Random Access Memories," Report RADC-TR-76-79, Rome Air Development Center, Griffiss Air Force Base, NY, March 1976.

6. Cocking, J., "RAM Test Patterns and Test Strategy," in Dig. Semiconductor Test Symposium, Cherry Hill, NJ, IEEE Computer Society, pp. 1-8, October 1975.

7. Barraclough, W., Chiang, A. C. L., and Sohl, W., "Techniques for Testing the Microcomputer Family," Proc. IEEE, Vol. 64, pp. 943-950, June 1976.

8. Springer, J., "Making Sense out of Delay Specs in Semiconductor Memories," Electronics, pp. 82-88, October 25, 1971

9. Brown, J. R., Jr., "1103 Semiconductor Memory Device Pattern Sensitivity and Error Modes," in Dig. Semiconductor Test Symposium, Cherry Hill, NJ, IEEE Computer Society, pp. 63-76, October 1973.

10. Webb, C. and Richardson, B., "Pattern Sensitivity in a 4096 Bit RAM," in Dig. Semiconductor Test Symposium, Cherry Hill, NJ, IEEE Computer Society, pp. 33-52, November 1974.

11. Richardson, W. S., "Diagnostic Testing of MOS Random Access Memory," in Dig. Semiconductor Test Symposium, Cherry Hill, NJ, IEEE Computer Society, pp. 7-20, October 1973.

12. Huston, R. E., "Testing Semiconductor Memories," in Dig. Semiconductor Test Symposium, Cherry Hill, NJ, IEEE Computer Society, pp. 27-62, October 1973.